CASIO®

CASIO PB-700 EASY TRIP TO

CASIO PB-700

EASY TRIP TO BASIC

Here's how Casio opens up the complex world of programming to help you in more ways than you could imagine.

CASIO®

CASIO PB-700

EASY TRIP TO BASIC

Here's how Casio opens up the complex world of programming to help you in more ways than you could imagine.

CASIO®

INTRODUCTION

The PB-700, a practical personal computer that can be easily carried anywhere in your bag, provides the same functions as a conventional table top personal computer because it has a large RAM capacity which can be expanded to a maximum of 16K bytes. In other words, it allows about 15,000 characters to be stored which provides an adequate capacity for regular personal programming.

RAM (Random Access Memory) is a memory device in which you can freely read and write programs and data. To freely write data into RAM, process the data in response to your purpose, and recall only necessary data, first a program has to be written into this RAM which matches the application by using a programming language called BASIC.

A BASIC program is a procedure for performing work.

Since a computer is worthless without a program, this manual was prepared as an easy guide for customers who would like to learn the BASIC language so that the considerable functions of the PB-700 can be fully utilized.

PB-700 functions can be broadly expanded by connecting an optional plotter-printer with cassette interface which allows a mini plotter-printer to be used for quickly preparing beautiful 4-color graphs and tables. The method for accomplishing this is roughly explained in this manual. Because the PB-700 is a practical personal computer, it provides excellent print output.

It is certain that you will like the PB-700 more and more as you become used to it. The purpose of this manual is to create an opportunity for as many readers as possible to become familiar with the PB-700.

CONTENTS

CHAPTER 1 GENERAL GUIDE

1-1 PRIOR TO OPERATION 10
1-2 SYSTEM CONFIGURATION AND CONNECTIONS 11
1-3 BATTERY MAINTENANCE 12
1-4 RAM EXPANSION PACK (OPTIONAL) 14
1-5 EACH SECTION'S NOMENCLATURE AND OPERATION 15
1-6 TEST OPERATION 16

CHAPTER 2 KEY OPERATION AND DISPLAY

2-1 KEY FUNCTIONS IN THE DIRECT MODE 18
2-2 KEY FUNCTIONS IN THE SHIFT MODE 19
2-3 CAPS MODE KEY FUNCTIONS 19
2-4 EDITING AND SPECIAL KEY FUNCTIONS 20
2-5 CALCULATION FUNCTIONS 21
2-6 VARIABLES 24
2-7 DISPLAY SCREEN 26
2-8 NUMBER OF BYTES USED FOR VARIABLES 27

CHAPTER 3 "BASIC" REFERENCE

3-1 BASIC 30
3-2 USING THE KEYS 31
3-3 VARIABLES AND ASSIGNMENT 33
3-4 USING VARIABLES 35
3-5 PROGRAM ENTRY 36
3-6 BASIC PROGRAMMING [1] 37
3-7 BASIC PROGRAMMING [2] 40

3-8 PROGRAM EXECUTION 46
3-9 DISPLAY SCREEN CONFIGURATION 48
3-10 REPEATING A SPECIFIED NUMBER OF TIMES 50
3-11 SUM TOTAL PROGRAMMING 54
3-12 CHARACTER VARIABLES 57
3-13 WHAT IS A DIMENSION? 59
3-14 NUMERICAL ARRAY VARIABLES 62
3-15 NUMERICAL ARRAY PROGRAMMING 67
3-16 CHARACTER ARRAY VARIABLES 74
3-17 COMBINATION OF CHARACTER ARRAYS AND NUMERICAL ARRAYS 78
3-18 HOW TO MAKE A FLOWCHART 82
3-19 PB-700 GRAPHIC FUNCTIONS 86
3-20 GRAPHIC COMMANDS AND SCREEN COORDINATES 87
3-21 DRAWING A CURVE 92
3-22 DRAWING A LINE GRAPH 94
3-23 PREPARATION FOR DRAWING A BAR GRAPH 96
3-24 TWO EXAMPLES OF BAR GRAPH PROGRAMS 98
3-25 ANIMATION DRAWING 101
3-26 GAME APPLICATIONS 105
3-27 DRAWING A PATTERN WITH THE PLOTTER-PRINTER 108
3-28 DRAWING A STRAIGHT LINE 111
3-29 RECTANGLE, CIRCLE AND COLOR SPECIFICATION 113
3-30 TECHNIQUE FOR DRAWING A GRAPH 116
3-31 HOW TO WRITE GRAPHIC CHARACTERS 121
3-32 DRAWING A SINE CURVE 127
Preface to Chapter 4 129

CHAPTER 4 COMMAND REFERENCE

COMMAND, STATEMENT AND BUILT-IN FUNCTION TABLE 132
4-1 MANUAL COMMANDS 136
CONT 136
DELETE 138
EDIT 141
LIST/LLIST 145
LOAD 149
NEW/NEW ALL 153
PASS 155
PROG 157
RUN 158
SAVE 159
SYSTEM 162
VERIFY 163
4-2 PROGRAM COMMANDS 164
ANGLE 164
BEEP 165
CHAIN 166
CLEAR 168
CLS 169
DIM 170
DRAW/DRAWC 175
END 178
ERASE 179
FOR ~ TO ~ STEP/NEXT 180
GET 185
GOSUB/RETURN 188
GOTO 192
IF ~ THEN ~ ELSE 194

INPUT 197
LET 203
LOCATE 204
PRINT/LPRINT 205
PUT 210
READ/DATA/RESTORE 212
REM 217
STOP 218
TRON/TROFF 220
4-3 NUMERICAL FUNCTIONS 222
SIN 222
COS 225
TAN 226
ASN, ACS, ATN 227
SQR 229
LOG, LGT 230
EXP 233
ABS 235
INT 237
FRAC 239
SGN 241
ROUND 243
PI 245
RND 246
4-4 CHARACTER FUNCTIONS 248
ASC 248
CHR$ 250
VAL 252
STR$ 255

LEFT$ 257
RIGHT$ 258
MID$ 259
LEN 261
INKEY$ 262
4-5 DISPLAY FUNCTIONS 265
TAB 265
USING 267
POINT 271

CHAPTER 5 PROGRAM LIBRARY

STOCK PRICE MANAGEMENT AND PROPER SELLING/BUYING PRICES 274
TELEPHONE DIRECTORY 282
CROSS TOTAL 288
VARIOUS GRAPH MAKING 296
WORLD STANDARD TIME 303

CHAPTER 6 REFERENCE MATERIAL

6-1 PB-700 COMMAND TABLE 308
6-2 ERROR MESSAGE TABLE 317
6-3 CHARACTER CODE TABLE 321
SPECIFICATIONS 322

OUTLINE

Because the PB-700 has many functional characteristics, it is not easy for the personal computer beginner to learn them all at once in a short period of time.

As a rule, the only learning method that applies is understanding one item at a time and taking the time to master each item.

The skill achieved in using a personal computer is considered to be in proportion to the period of time that the keys are used.

Since this manual has a configuration and descriptions that allow the personal computer beginner to easily understand it, many practice programs are prepared as part of an instruction process that allows the reader to learn the BASIC language while performing key operations. To provide correct PB-700 operation, an outline of its features and basic handling precautions are covered in Chapter 1, while the functions of each key and the display screen are covered in Chapter 2 before you learn BASIC.

The main functions of a computer are to store and compute as a matter of course. How should a BASIC language program be prepared to store considerable data in the PB-700 and recall it freely when necessary? To answer this question, a detailed explanation of "Array Programming" is provided in Chapter 3.

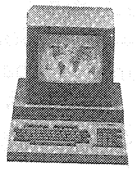
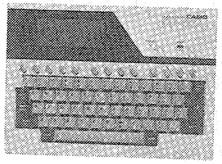
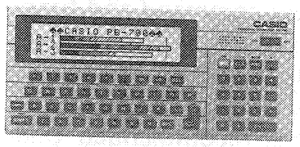
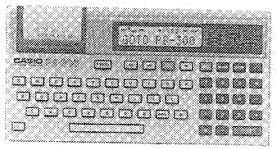
The essentials of BASIC programming are explained in Chapter 3 for the BASIC beginner including graphic programming for fully utilizing the large display of the PB-700, as well as the applications of the optional plotter-printer with cassette interface for expanding PB-700 functions.

The user that has already mastered BASIC should read Chapter 4 instead of Chapter 3. In Chapter 4, the utilization of many command and functions of the PB-700 are explained in detail as much as possible. It is not efficient for the personal computer beginner to learn the commands in this chapter serially from the beginning. Since only 10 BASIC commands are necessary from a configuration viewpoint, it is recommended that these commands be selected and learned so that full understanding is achieved.

In Chapter 5, representative program examples are provided for actual PB-700 applications. It is our desire that the PB-700 is effectively utilized by referring to these program examples and rearranging the programs so they can be easily used.

THE PB-700 AND OTHER CASIO PERSONAL COMPUTERS

The PB-700 is a high class PB Series computer which is provided with considerable BASIC commands as well as a large display window and memory capacity that are adequate for small to medium work requirements as shown below.
Also, the optional plotter-printer with cassette interface can expand PB-700 functions for graph preparation and hobby use as an attractive feature.

Model	RAM Capacity	Characteristics
FP-1100 	Main 64K bytes Sub 48K bytes	● Dual CPUs, excellent computing precision (decimal arithmetic), double-double precision. ● Considerable Graphic functions.
FP-1000	Main 64K bytes Sub 16K bytes	● Basic Machine of the FP-1100
FP-200 	32K bytes Maximum	● Handy Computer ● 20 Digit, 8 Line Display Panel ● CETL, an easy table language, is provided.
PB-700 	16K bytes Maximum	● 20 Digit, 4 Line Display Panel ● Plotter-Printer with Cassette Interface ● Powerful BASIC equivalent to the FP-200
PB-300/FX-802P 	About 2.3K bytes	● Built-in Printer
FX-700P	About 2.3K bytes	● With Expanded Memory
PB-100	About 2.3K bytes Maximum	● Handy BASIC Guide Machine

CHAPTER 1

GENERAL GUIDE

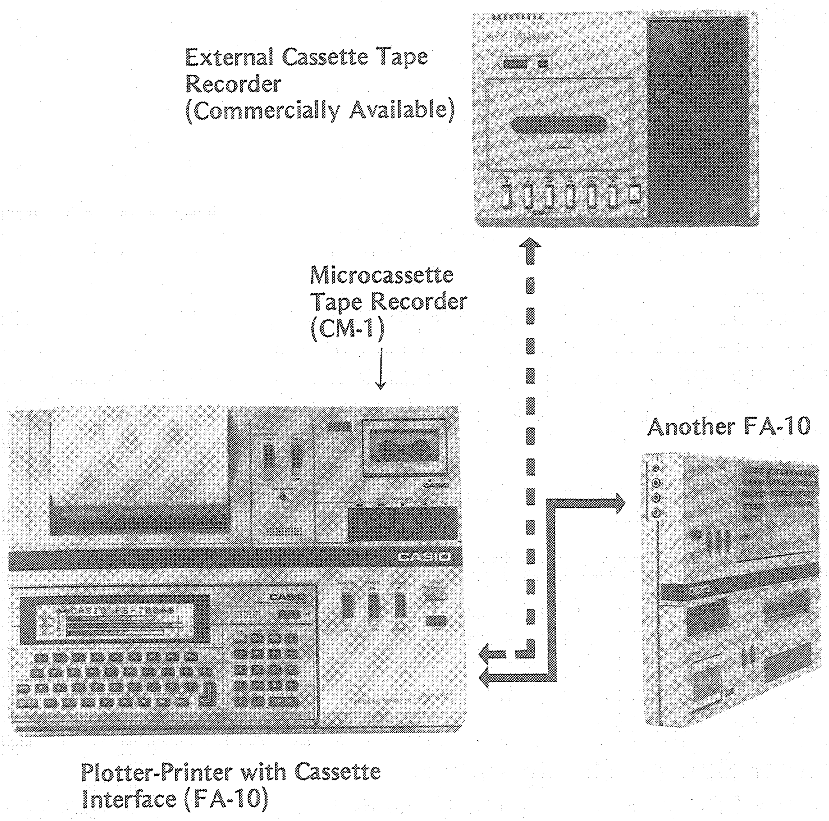
1-1 PRIOR TO OPERATION

This computer was delivered to you through CASIO's strict testing process, high level electronics technology, and strict quality control.
To ensure a long life for your computer, please observe the following precautions.

■ Utilization Precautions

- Since this computer is constructed with precision electronic components, never attempt to disassemble it. Also special care should be taken so that it is not damaged by bending or dropping. For example, do not carry it in your hip pocket.
- Since the FA-10 (plotter-printer with cassette interface) should only be connected, do not connect other equipment to the connector.
- Avoid extreme temperature variations. Also, be especially careful to avoid high temperature locations with high humidity or a lot of dust. However, if the ambient temperature is too low, the display response speed may become slow or there may be no display. When normal temperature conditions are restored, the computer operation will become normal.
- To keep the computer clean, wipe its surface with a soft, dry cloth or one dampened with a neutral detergent.
- When a malfunction occurs, contact the store where it was purchased or a nearby dealer.
- Before seeking service, please read this manual again and check the power supply as well as the program for logic errors.

1-2 SYSTEM CONFIGURATION AND CONNECTIONS


External Cassette Tape Recorder (Commercially Available)
Microcassette Tape Recorder (CM-1)
Another FA-10
Plotter-Printer with Cassette Interface (FA-10)

The PB-700 can be connected to a commercially available cassette tape recorder through the FA-10 Plotter-Printer with Cassette Interface. Also, program data transfer can be performed with another PB-700 through another FA-10.
Be sure to turn off the power prior to performing connections.
After installing or removing the RAM Expansion Pack OR-4, be sure to enter NEW ALL ⏎ with the power switch ON.

1-3 BATTERY MAINTENANCE

Battery Insertion

Turn the PB-700 power off, then turn over the main frame and open the battery compartment lid by sliding it (Fig. 1).
Install the batteries so that the minus side is at the spring side of the main frame (Fig. 2).
Be sure to insert them with correct polarities (+, —) to prevent the possibility of battery burst.
Four AA size batteries are used. Do not use old and new batteries together because the battery life will be considerably shortened.
If it is not used for a long period of time, store the main frame with the batteries removed.
Do not throw old batteries into a fire because they may burst.

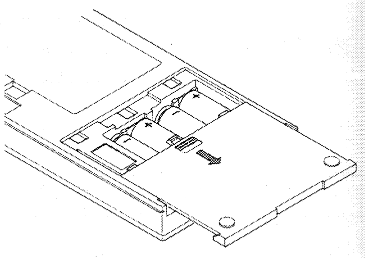


Fig. 1

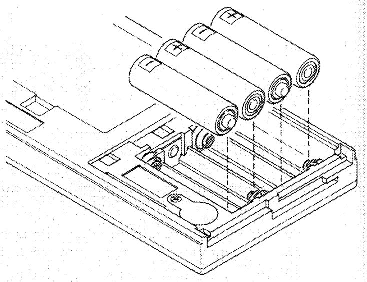


Fig. 2

Battery Replacement Period

When the buzzer sound produced by the BEEP command becomes small, or the display becomes blank, replace the batteries with new ones. (For battery specifications, see page 323.)

Power Source Configuration

Since the PB-700 power source is divided into a main power source and a sub power source for the RAM backup (Fig. 3), a program or data in the mainframe are not lost during main power source or sub power source battery replacement. A program or data are only lost when the main and sub power sources are removed at the same time.

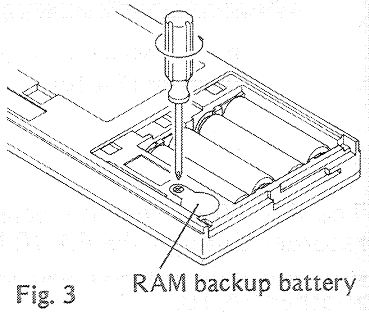


Fig. 3

- * Be sure to replace the main batteries every 18 months and the RAM backup battery every 2 years regardless of their use in order to prevent the chance of malfunction due to battery leakage.
- * With the main batteries removed, the sub battery protects RAM for approx. 10 months when RAM capacity is standard 4K bytes (approx. 2.5 months when RAM capacity is expanded to 16K bytes).

- * When both the main batteries and the RAM backup battery are replaced, be sure to press NEW ALL ↵ in sequence.
- * Keep the battery away from children. If it is swallowed, contact your doctor immediately.

Auto Power Off

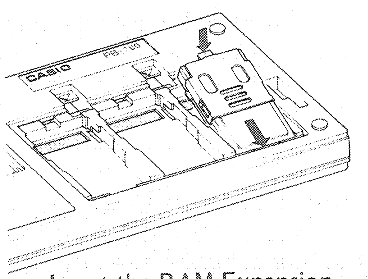
This is an automatic energy-saving function which prevents power consumption when you forget to turn off the power. Approximately 8 minutes after the last key operation (except during program execution), power will go off automatically.
Power can be resumed by pressing the ON/BRK key or turning the power switch off and then on again.

Low-voltage Detection Feature

The PB-700 has a Low-voltage Detection feature for protecting RAM contents when the main batteries become exhausted and their voltage goes below a certain level.
When the voltage becomes low, the whole display becomes blank or the display during program execution blinks, and the PB-700 is no longer operable. The batteries must then be replaced at once. If the power is turned on with exhausted batteries installed, RAM contents may change.

1-4 RAM EXPANSION PACK (OPTIONAL)

Fig. 1 After turning off the power . . . .



Insert the RAM Expansion Pack as shown above.

Fig. 2 Slide the holder in the direction of the arrow . . .

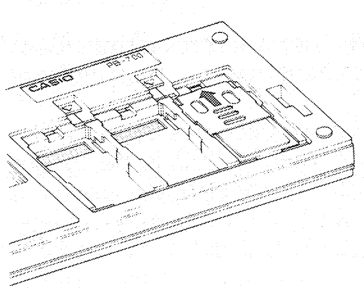
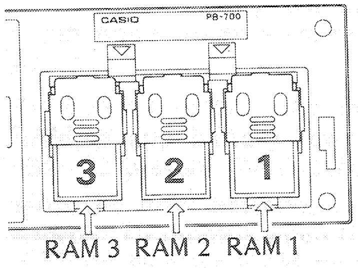


Fig. 3 RAM Expansion Pack insertion sequence . . .



If the RAM capacity of the PB-700 is not expanded, the basic capacity is 4K bytes. The RAM capacity can be expanded up to a maximum of 16K bytes by installing OR-4 RAM Expansion Packs.
One RAM Expansion Pack provides 4K bytes.
They are installed by the following procedures.

(1) Turn off the power of the PB-700.
(2) Turn over the PB-700 and remove the RAM Box Cover by holding its two catches.
(3) Insert the OR-4 RAM Expansion Pack into the far right position by holding its edge (Fig. 1)
Since RAM is very sensitive to static electricity on a human, be careful not to touch the OR-4 terminals.
(4) Slide the holder into a locked position while pressing it gently (Fig. 2).
(5) Install the required number of RAM Expansion Packs, then place the RAM Box Cover and turn the PB-700 power on.
(6) Enter NEW ALL ↵ . Then confirm the RAM capacity by entering SHIFT SYSTEM S ↵ .
(7) Install the RAM Expansion Packs with the sequence of 1—2—3 (Fig. 3).
If a RAM Expansion Pack is installed in position 2 by skipping position 1, correct functions cannot be performed.

1-5 EACH SECTION'S NOMENCLATURE AND OPERATION

- **Display contrast control** . . . Adjust it so that the display can be easily read.
- **Alphabet keys** . . . . When pressed, capital letters are displayed. When the CAPS key and Alphabet keys are pressed at the same time, small letters are displayed. Also, Commands or Symbols printed above each key can be displayed by pressing the SHIFT key and Alphabet keys at the same time (one key commands).

Example:

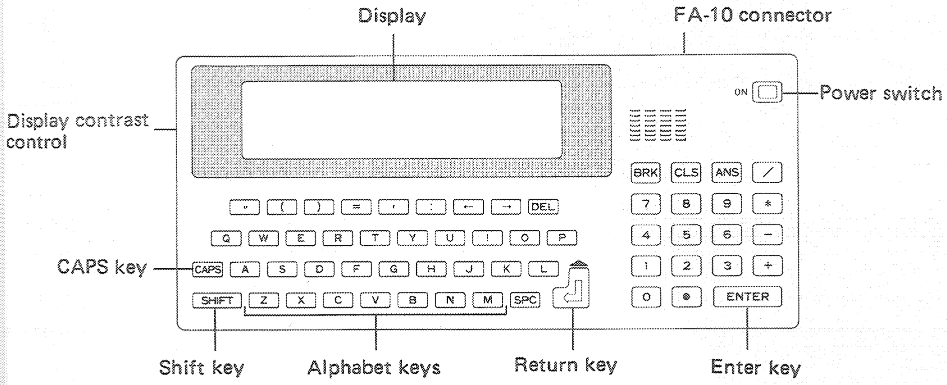
SYSTEM · · · SHIFT mode
S · · · DIRECT mode

* Use the CHR$ function for symbols and characters which cannot be input by directly pressing a key. (See page 250.)

- **Return key (↵)** . . . . . . Used to perform BASIC command or program input and output, and to execute an instruction for program data input and output.
- **Enter key (ENTER)** . . . . . . When a manual calculation is performed, it is used to execute an instruction.

Example: 1 2 / 4 ENTER → 3

■ See "2-4 Editing and Special Key Functions" in Chapter 2 for the detailed use of each key.



1-6 TEST OPERATION

This is a demonstration program to show (though roughly) the functions of PB-700. If you are not sure that you can input the program correctly, refer to Chapters 2 and 3.

By this program, you can see various functions of the display screen, BEEP command and the execution speeds of the graphic commands.

```
  10 CLS
  20 LOCATE 3,1:PRINT "PB-700 TESTING"
  30 FOR A=31 TO 0 STEP -1
  40 DRAW(0,A)-(159,A)
  50 NEXT A
  60 LOCATE 5,1:PRINT "BEEP SOUND"
  70 FOR A=0 TO 9
  80 BEEP 1:BEEP 1:BEEP 0
  90 NEXT A
 100 CLS
 110 FOR A=33 TO 255
 120 PRINT CHR$(A);
 130 NEXT A
 140 FOR A=0 TO 20
 150 NEXT A
 160 CLS
 170 FOR A=1 TO 16
 180 DRAW(A+2,16-A)-(A*3,16-A)-(A*3,15+
A)-(A+2,15+A)-(A+2,16-A)
 190 NEXT A
 200 FOR A=0 TO 200
 210 NEXT A
```

CHAPTER 2

KEY OPERATION AND DISPLAY

2-1 KEY FUNCTIONS IN THE DIRECT MODE

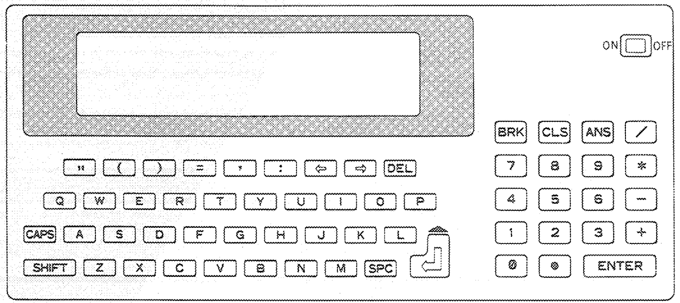
When a key entry is made in the direct mode, the input of the character or function on the key top is accomplished.

A — Z	Capital alphabetic characters
SPC	Space (blank)
", =, (,), ,, :	Symbols
0 — 9	Numbers
.	Decimal point
+, −, *, /	Calculation command symbols
CLS	Display (screen) clear
DEL	Character delete
BRK	Break (execution halt)
ENTER	Manual calculation execution
↵	Program input and execution
⇦, ⇨	Cursor movement (left, right)
SHIFT	SHIFT mode designation
CAPS	CAPS mode designation
ANS	Answer key (Last calculation result access)

■ Number of characters in one statement

The maximum number of characters that can be entered in a calculation formula during manual calculation or in one line during BASIC programming is 79 characters.

Key Functions In the Direct Mode



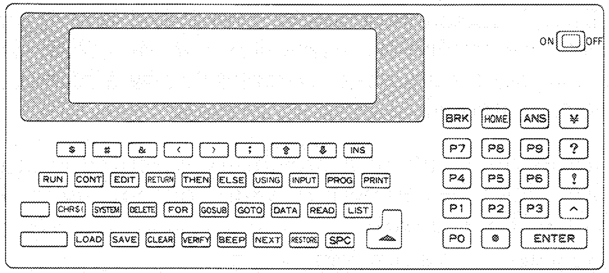
ON OFF
BRK CLS ANS /
7 8 9 *
4 5 6 −
1 2 3 +
0 . ENTER
" ( ) = , : ⇦ ⇨ DEL
Q W E R T Y U I O P
CAPS A S D F G H J K L
SHIFT Z X C V B N M SPC

2-2 KEY FUNCTIONS IN THE SHIFT MODE

This function occurs when the SHIFT key and another key are pressed at the same time. The brown characters, symbols, etc. printed above each key are entered.
26 different one key commands are provided.

&, #, <, >, $, ;, ¥, ?, !, ^	Symbols
.	Decimal point
INS	Character, symbol insertion
P0 — P9	Program area designation
▲	Previous line display in the EDIT mode (see page 141.)
⇧, ⇩	Cursor movement (up, down)
HOME	Cursor movement (to the beginning of a statement)

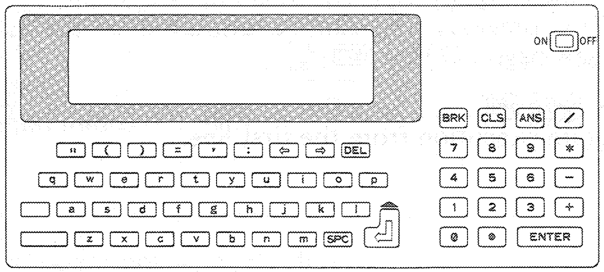
Key Functions In the Shift Mode



ON OFF
BRK HOME ANS ¥
P7 P8 P9 ?
P4 P5 P6 !
P1 P2 P3 ^
P0 . ENTER
$ # & < > ; ⇧ ⇩ INS
RUN CONT EDIT RETURN THEN ELSE USING INPUT PROG PRINT
CHR$(SYSTEM DELETE FOR GOSUB GOTO DATA READ LIST
LOAD SAVE CLEAR VERIFY BEEP NEXT RESTORE SPC

2-3 CAPS MODE KEY FUNCTIONS

This is provided by pressing the CAPS key and another key at the same time.



ON OFF
BRK CLS ANS /
7 8 9 *
4 5 6 −
1 2 3 +
0 . ENTER
" ( ) = , : ⇦ ⇨ DEL
q w e r t y u i o p
a s d f g h j k l
z x c v b n m SPC

2-4 EDITING AND SPECIAL KEY FUNCTIONS

(1) SPC ..... (SPACE) Blank entry in all key modes.

(2) HOME CLS ..... (Clear screen/Home) Clears the display and moves the cursor to the top left of the display.
SHIFT HOME only moves the cursor to the beginning of a statement while the display remains as it is.

(3) INS DEL ..... (Delete/Insert) Deletes a character or symbol where the cursor is located, and moves the character or symbol at the right of the cursor to the left.
SHIFT INS moves the character or symbol at the right of the cursor to the right and inserts a blank.

(4) ON BRK ..... (Break) Halts calculation and program execution. Also used to turn the power on when auto power off occurs.

(5) ↑⇦, ↓⇨... Moves the cursor left, right, up, down. The repeat function provides only left, right movement.

(6) ENTER ... (Enter) Executes manual calculation. During key input wait in an INPUT statement, it functions as ↵. An assignment statement, command, and a statement cannot be executed with ENTER .

(7) STOP ANS ..... Stores the result of a previously executed manual calculation and the numerical value output by a PRINT or LPRINT statement.

Example: 3.4 ∗ 5 ENTER → 17
5.8 ∗ 3 − ANS ENTER → 0.4

During program execution it provides a program stop function. The program can be continued by using a CONT statement (see STOP command).

(8) ▲↵ ..... (Return/Line back) Executes program input and commands. Manual operation cannot be executed.
The previous line can be displayed in the EDIT mode (see page 141) by SHIFT ▲.

(9) SHIFT P0 — SHIFT P9 .... Specifies a program area and causes program execution from the first line.

2-5 CALCULATION FUNCTIONS

Calculation Precision and Functions

All internal calculations are performed by a 12 digit mantissa (+ 2 digit exponent). Since decimal operation is used, high precision calculations can be performed.
Manual calculations are executed with the ENTER key.
Calculation results are displayed with a 10 digit mantissa (+ 2 digit exponent). In this case, the 11th mantissa digit is rounded to the nearest whole number.

Operator Functions

^	Power
+, −	Addition, Subtraction
∗, /	Multiplication, division
MOD	Remainder calculation (If the numerical value includes a fraction, the fraction is discarded by this operation.)

The calculation range is as follows.

(1) Division by 0 causes an MA error.
(2) When overflow occurs (i.e. when a result exceeds the calculation range), it causes an OV error.
(3) Power range

$0 \wedge 0 \longrightarrow$ MA error
$(\pm x) \wedge 0 \longrightarrow 1$
$0 \wedge y \longrightarrow 0$
$0 \wedge (-y) \longrightarrow$ MA error
$(-x) \wedge (\pm y) \longrightarrow$ OK only when y is an integer. Otherwise it causes an MA error.
* Where $x > 0, y > 0$.

Calculation Priority Sequence

Calculations are executed in the following sequence.

1. Elements in parentheses
2. Functions
3. Powers (^)
4. Plus (+) and minus (−) signs
5. ∗, /
6. MOD
7. +, −
8. Relational operators (=, >, <, etc.)

Calculation Methods

Formats

1. $\frac{X+Y}{2}$ → (X+Y)/2
2. $X^2+2\times Y+Y^2$ → X^2+2*X*Y+Y^2
3. $-Y^2$ → -Y^2
4. $(-Y)^2$ → (-Y)^2
5. $(X^Y)^2$ → X^Y^2
6. X^{Y^2} → X^(Y^2)
7. Remainder of $\frac{X}{Y}$ → X MOD Y

Examples

1. 0.5^0 → 1
2. -0.5^0 → -1
3. (-0.5)^0 → 1
4. 0.5^2 → 0.25
5. 0.5^-2 → 4
6. (-0.5)^-2 → 4
7. 0.5^0.5 → 0.7071067812
8. (-0.5)^0.5 → MA error
9. 2^-0.5 → 0.7071067812
10. (-2)^-0.5 → MA error
11. 10 MOD 6 → 4
12. -10 MOD 6 → -4
13. 10 MOD-6 → 4
14. -10 MOD-6 → -4

Relational Operators

Relational operators can only be used in an IF statement (see page 194).

=	Equal
<>, ><	Not equal
<	Smaller
>	Larger
=>, >=	Either larger than or equal to.
=<, <=	Either smaller than or equal to.

Examples: A + B < > 0. . . The result of A + B is not equal to 0.
A$< >"Y" . . . The content of A$ is not equal to "Y".
A$ = CHR$(84) + CHR$(79) + CHR$(77) . . . A$ is equal to "TOM".
CHR$(67) > CHR$(N) . . . CHR$(67), which is C, is larger than CHR$(N) in the Character Code Table (page 321).

Operations Using Variables

The content of a variable can be confirmed with the ENTER key.

Example: A ENTER → 0, AN ENTER → 0 (when used as a registered variable)

When a numerical value is entered in a variable, the contents of the variable are as follows.

Single-precision: 13th mantissa digit and after are discarded (12 digits). (Single-precision is the calculation precision in a normal state.)

Half-precision: 6th mantissa digit and after are discarded (5 digits). (Half-precision is 5 digit numerical values realized by specifying an array variable (!).) (It can only be specified in an array variable.)
(See page 65.)
(The mantissa is the input numerical value.)

Half-precision computation is convenient.

In many cases, five digits are adequate for computations except for technical computations. For example, storage of a test result, the component ratio (%), product No., unit price, quantity, etc. can be reduced to one-half that for single-precision by using half-precision if they are kept within five digits which allows more data to be stored in RAM.
Although it is generally necessary to use 8 bytes for 1 numerical variable (in single-precision), only 4 bytes are required when half-precision is used.

2-6 VARIABLES

Kinds of Variables

The kinds of PB-700 variables are as follows.

(1) **Numerical variables**
   Numerical fixed variables (up to 12 digits)
   Numerical registered variables (up to 12 digits)
   Numerical array variables (Half-precision numerical array: up to 5 digits, Single-precision numerical array: up to 12 digits).
   * The number of digits shown above is the internal calculation digits.

(2) **Character variables**
   Character fixed variables (up to 7 characters)
   Character registered variables (up to 16 characters)
   Character array variables (Fixed-length character array: up to 16 characters, Defined-length character array: 0 to 79 characters).

Fixed-length Character Array ..... Indicates when the character variables A$ to Z$ are used in an array variable.

Example: DIM A$ (20, 10)

Defined-length Character Array ... Indicates when the array names A$ to Z$ are used in an array variable, and when a specification of the number of characters is performed.

Example: DIM A$(9, 9) * 79

Maximum value from 0 – 79 to be used.

Fixed Variables (A–Z, A$–Z$)

Memory where numerical values or characters are stored has 26 kinds of Fixed Variables which are A–Z or A$–Z$. Numerical fixed variables and character fixed variables with identical names cannot be used together at the same time. If identical variable names are used, a UV error will occur.

Incorrect Usage: 10 PRINT A ; A$ → UV error

Registered Variables

Variable names with two characters that consist of a capital alphabetical character and a numerical value, as well as Fixed variables, can be used. If a variable name is defined with three characters or more, an SN error will occur during execution.

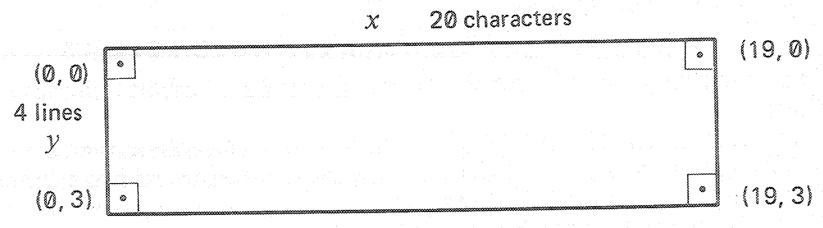
Examples: AB, X1, Y1, X2, Y2, AZ$, AA$, B1$, Z9$

- The beginning of a variable name must be a capital alphabetical character.
- A reserved word (IF, TO, PI, etc.) cannot be a variable name.
- A 12 digit mantissa + a 2 digit exponent can be stored in a numerical registered variable (AB, X1, etc.).
- Up to 16 characters can be stored in a character registered variable.
- 40 registered variables including the array variable names can be used. If it exceeds 40 variables, a VA error will occur that will halt execution, then the variable name shall be cleared with CLEAR or ERASE.
- A registered variable name can be recalled by executing LISTV. A numerical registered variable uses 8 bytes and a character registered variable uses 17 bytes.

2-7 DISPLAY SCREEN

■ Character Coordinates

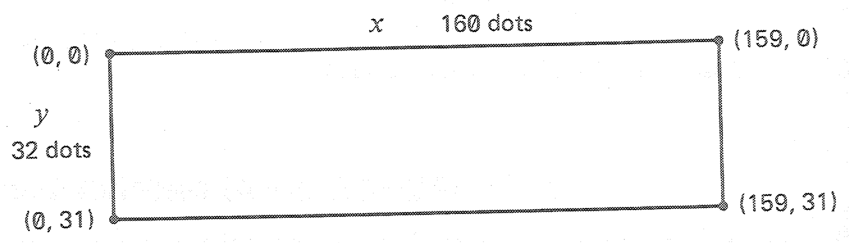
Twenty characters horizontally and four character lines vertically fit in the display window (LCD). Character locations are expressed by a LOCATE statement with the following coordinates.



LOCATE (X, Y) . . . . See LOCATE.
Based on the coordinates mentioned above, 222 characters in the character code table (see page 321) can be displayed.

■ Graphic Coordinates

The display can be specified with dot units based on the coordinates shown below. Dot locations are expressed by a DRAW or DRAWC statement which allows dots and straight lines to be drawn or erased.



DRAW (X, Y) . . . . See DRAW, DRAWC.
Also, whether a dot on graphic coordinates is lit or not can be checked by using the POINT function.
POINT (X, Y) . . . . See POINT.

2-8 NUMBER OF BYTES USED FOR VARIABLES

In regard to all variables except a fixed variable, when data is assigned during program execution, the remaining RAM capacity is reduced. The required number of bytes per variable is as follows. Since the fixed variables (A — Z) are provided separately from the freely used RAM area, byte computation is not required.

Registered Variables		Array Variables	
Variable	Number of Bytes Used	Variable	Number of Bytes Used
Numerical variable	8 bytes	Numerical variable (Half-precision)	4 bytes
Character variable	17 bytes	Numerical variable (Single-precision)	8 bytes
		Character variable (Fixed-length)	17 bytes
		Character variable (Defined-length)	2—80 bytes (1—79 characters)

CHAPTER 3

“BASIC” REFERENCE

3-1 BASIC

You have probably heard or seen the word BASIC.
Although BASIC is an acronym for Beginner's All-purpose Symbolic Instruction Code, it actually could be interpreted as a Basic language, as the word means.
Whatever its interpretation is, BASIC is a sophisticated programming language that beginners can easily learn, and is an appropriate computer guide as the word indicates. In this connection, a sophisticated program language is a language which allows program preparation with terminology that is close to a daily language.
In addition to BASIC, there are many other sophisticated program languages such as FORTRAN, COBOL, PASCAL, PL/I, etc. BASIC was invented at a U.S. university in 1964 to provide a language for a large computer.
BASIC became more popular for personal computers than any other language because of its convenience in allowing programs to be prepared with computer and human interaction.
The advantage of BASIC is that interactive programming allows programs to be prepared by performing many different trials.
Since the PB-700 can be carried anywhere, it is certain that you can gradually learn BASIC programming while you operate the computer every day.

3-2 USING THE KEYS

Although the PB-700 has a large data processing capacity, and can perform complicated numerical calculations, it can also be used to perform manual calculations without using a program.

To get used to the PB-700, let's start with a very simple operation.
Turn the power on and the following is displayed.

```
Ready P0
```

(This means that the program area is specified to No. 0.)

When the PB-700 is used as a calculator, the numerical keys (ten keys) on the right side of the main frame are mainly used. [×], [÷], and [=] are not included with the ten keys like a normal calculator. Although [=] is located on the left side, it cannot be substituted for a [=] key on a normal calculator.
The [*] and [/] keys are used for [×] and [÷] respectively, while the [ENTER] key functions as an [=] key.
Let's try 1+2. When you enter [1] [+] [2], is the following displayed?

```
Ready  P0
1 + 2 _
      └ Cursor
```

If you make a mistake, move the Cursor to the location of the mistake by using the [⇦] and [⇨] keys, and input the correct value.
Next, when you press the [ENTER] key, the answer is displayed as follows.

```
1 + 2
  3

_
```

Since the PB-700 provides many different functions such as powers, trigonometric functions, inverse trigonometric functions, logarithmic functions, etc. besides the four calculation functions (+, –, *, /), more complex calculations can be performed by changing the format slightly if necessary.

[Numerical Expression]		**[Input Format]**
5 × 6 ÷ 2	➡	5*6/2 ENTER
6.5^2	➡	6.5^2 ENTER
$\frac{\text{SIN } 30^\circ + \text{COS } 60^\circ}{\text{TAN } 45^\circ}$	➡	(SIN 30 + COS 60)/ TAN 45 ENTER

3-3 VARIABLES AND ASSIGNMENT

Now let's try another calculation.

$500000 \times (1 + 0.07)^{10}$, $800000 \times (1 + 0.07)^{10}$

In these expressions, an amount with the interest added after 10 years is calculated based on a compound interest method. What is the simplest way to perform these two calculations? Once a calculation expression is input, it is partially used repeatedly. So this part is stored in a memory and recalled when it is needed.
Therefore,

Enter A = (1 + 0.07)^10 ⏎ and change the previous two expressions as

A * 500000 ENTER , A * 800000 ENTER

then the calculation becomes easier to perform.
The value of $(1 + 0.07)^{10}$ is stored in A. This A is called a variable in the program.
To store a certain numerical value in variable A, the following operation is performed.

A = 176 ⏎ (Assign 176 to A)

(Left side) (Right side)

Since an assignment is made to store the right side (176) in the left side (A), 176 is assigned to A in this example. Here the assignment instruction is performed by using " = ". To confirm that 176 is entered, enter A ENTER which means to display the content of variable A. Is 176 displayed?
Since this is very important in understanding BASIC, please make sure that you understand this correctly.
" = " is the assignment instruction, and does not mean equal as used in mathematics (except in a IF statement). For example,

A = A + 1 ⏎

means to assign the value of A + 1 to A. By assuming that 176 is stored in variable A, when this expression is executed by the computer, 177 is assigned to variable A. Enter A ENTER to confirm this.
Is the following displayed?

```
A
 177

— (Cursor)
```

■ How ENTER and ↵ Differ

Another important item is your understanding of the difference in the functions of the ENTER and ↵ keys. It should be noted that the ↵ key is used for A = 176 and the ENTER key is used to display the value of A in the previous operations.

The ENTER key is used the same as the keys on a normal calculator such as for displaying an answer. This is called manual calculation.

On the other hand, the ↵ (Return) key is used to execute the instructions of a BASIC program. For example, it is used to execute program input, to correct a certain part of a program, or to execute a BASIC instruction.

3-4 USING VARIABLES

An alphabetical character such as A to Z, or two characters (alphabetical character + one character) such as AA, and N1 are used as variable names. Variables with a numerical value assigned as in the previous section can be freely used in calculation expressions.

Now, let's practice. When you enter

A = 36 ↵
B = 12 ↵

36 is assigned to A, and 12 is assigned to B. Next, enter

A + B ENTER

since ENTER is used "To display an answer" in a manual calculation. If 48 is displayed, perform the next step.

A − B ENTER → 24
A ∗ B ENTER → 432
(4 + A) ∗ LOG B ENTER → 99.39626599

Fairly complex calculations can be performed by freely using variables as mentioned above. However, this method has calculation procedure limitations. Therefore, a program is needed. If a certain level of using variables freely has been reached, it is not too difficult to prepare a program.

■ Program Areas

To enter many programs into the PB-700 for selective use as required, different programs are written in 10 program areas from P0 to P9. When you turn the power on, "Ready P0" is displayed. A number following P indicates the program area. During computer operation, press the BRK key to determine which program area is used.

To change the program area, such as moving to the program area P1, enter

PROG 1 ↵ or SHIFT PROG O 1 ↵.

3-5 PROGRAM ENTRY

Before you understand programming, it is necessary to make correct program entries. First you have to know the alphabetical keys. However, you will become accustomed to them since they have the same layout as a typewriter. To input a program, perform the following operations.

[SHIFT] [PROG O] [0] [↵] → Specifies the program area P0.
[N] [E] [W] [↵] → Erases the program stored in P0.
[CLS] → Clears the screen.

[Program]	[Key Operation]
10 CLEAR	[1] [0] [SHIFT] [CLEAR C] [↵]
20 A=A+1	[2] [0] [A] [=] [A] [+] [1] [↵]
30 LOCATE 7,2	[3] [0] [L] [O] [C] [A] [T] [E] [7] [,] [2] [↵]
40 PRINT A	[4] [0] [SHIFT] [PRINT P] [A] [↵]
50 GOTO 20	[5] [0] [SHIFT] [GOTO H] [2] [0] [↵]

→ ⌴ indicates to press the two keys at the same time.

Press [CLS] [SHIFT] [RUN Q] [↵] or [SHIFT] [PO 0] . If correct entries were made, numerical characters are displayed at the center of the screen at a high speed such as 1, 2, 3 . . . .
If "SN error P0 — line No." is displayed to indicate an entry error, correct (debug) the specified line as follows.

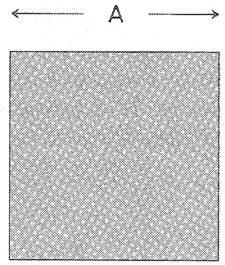
[SHIFT] [EDIT E] Line No. [↵]

The specified line is displayed when you perform this operation. Then move the cursor perform the correct input and press the [↵] key. The next line is displayed, and if no correction is required, press the [BRK] key, then press [SHIFT] [RUN Q] [↵] again. It should be noted that the EDIT mode is specified for program corrections.

3-6 BASIC PROGRAMMING [1]

Now, let's challenge BASIC programming.
A program that obtains the area of a square as shown on the right is prepared by using the following procedure.

(1) Requests input of the length of one side A.
(2) Computes the entered numerical value X (multiply) entered numerical value.
(3) Displays the computed numerical value.
(4) Returns to (1).



```
10  INPUT  A   ............... ①
20  B = A*A    ............... ②
30  PRINT  B   ............... ③
40  GOTO  10   ............... ④
```

Now, let's input this program by the following procedure. Press the keys correctly.

Power ON . . . . Ready P0 is displayed.
NEW[↵] . . . . [↵] is an arrow key located on the right below the alphabet keys.
10 [SHIFT] [INPUT I] A [↵] . . . . Press the [SHIFT] and [INPUT I] keys at the same time.
20 B=A*A [↵] . . . . [*] is located at the ten keys.
30 [SHIFT] [PRINT P] B [↵] . . . . Press the [SHIFT] and [PRINT P] keys at the same time.
40 [SHIFT] [GOTO H] 10 [↵] . . . . Press the [SHIFT] and [GOTO H] keys at the same time.

Next, let's execute this program with the following procedure.

[CLS] [SHIFT] [RUN Q] [↵] . . . . . [SHIFT] [PO 0] can also be used.

After this entry is made, ? _ is displayed. _ is called a cursor. Now, enter

8.5 ↵

and the next display should appear. If it does not appear, check if there is a program input mistake with SHIFT EDIT E ↵. Be careful not to make a mistake concerning the difference of 0 and O, and 1 and I.

```
RUN
? 8.5 . . . . . . . . . . . . . .  Value of one side is 8.5.
  72.25 . . . . . . . . . . . . .  Area is 72.25.
? _ (Cursor) . . . . . . . . . . . What is the value of one side ?
```

After confirming the execution of this program, let's analyze it.

10 INPUT A
(Make an entry) (to A)

"10" is a line number which indicates the program execution sequence. It is increased here by 10 for each line (to be explained later). INPUT means to make an entry, or in other words, " ? " is displayed to indicate that the computer is waiting for an entry. After an entry is made, the command stores it in a numerical variable.

20 B = A * A . . . . . . Assign the result of A * A to B.

Line 20 computes the area. The result of multiplying the entered numerical value by itself to provide the area of a square is assigned to B.

30 PRINT B
(Display) (B)

Line 30 displays the area. PRINT is used as a "Display" instruction. This line provides the instruction to display the content of B.

40 GOTO 10
(Go) (to line 10)

GOTO is a command that means "Go to" the line with the number that follows it.

The basis for the commands (INPUT, PRINT, and GOTO) used in this program should be understood after this explanation. However, this program is somewhat imperfect because it does not indicate what the input for "?" is, and what kind of computation result is provided when this program is executed. Therefore, let's add something to this program with the following procedure.

BRK . . . . Ready P0 is displayed.
SHIFT EDIT E ↵ . . . . Press the SHIFT and EDIT E keys at the same time.

After the content of line 10 is displayed, press ⇐ to move the cursor to the location of A, then enter " A = " SHIFT ; A ↵.
The modification result is as follows.

10 INPUT "A=" ; A

A message with the characters inside " " is displayed.

When a message is inserted, a semicolon (;) is required before the variable. (Be careful not to enter a colon (:) by mistake.)

Next, line 20 is displayed. However, since this line is not changed, press ↵.
Line 30 is displayed next. Move the cursor to the location of B, and enter " A R E A SPC " SHIFT ; B ↵. The modified result is as follows.

30 PRINT "AREA ␣" ; B

Message is displayed inside " ".

B is displayed following a message.

Press the SPC key.

After this, line 40 is displayed. However, since this line is not modified, press ↵.
Now, let's execute the program.

CLS SHIFT RUN Q ↵ . . . . SHIFT P0 0 can be used instead.

Now this program is considerably improved compared with the previous program, isn't it ? This is because it asks "A=? " and displays "AREA 72.25" as the computation result.

3-7 BASIC PROGRAMMING [2]

The basis for BASIC is further understood by entering a program which uses basic commands. The following program provides multiples of specified numerical values from 0 to 200.

```
 10 REM MULTIPLE↵ ···· Press the ↵ key at the end of each line.
 20 A=0↵
 30 INPUT "NUMBER";N↵
 40 A=A+1↵
 50 B=N*A↵
 60 IF B>200 THEN 100↵
 70 PRINT B;↵
 80 INPUT " OK";C$↵
 90 GOTO 40↵
100 END↵
```

After completing the program input, press the EDIT key while pressing the SHIFT key. Next, the program that was input is displayed from the beginning by ↵. If there is a mistake, move the cursor to correct it. Is the following displayed ?

To provide an understanding of the basic configuration for a BASIC program, the following explanation is provided. Press the ↵ key when requested.

[1] 10 REM MULTIPLE **See page 217.**

Line number Command Label

A BASIC program consists of line numbers, instructions (program instructions or part of an instruction), and variables or expressions that use variables.

Since line 10 is a REM statement which provides a comment statement, a comment called a label is provided instead of a variable which indicates this program is a multiple program. It is a kind of index. The content after REM is not executed. Now, press the ↵ key.

[2] 20 A=0

Line number Numerical variable (Fixed variable)

On line 20, 0 is assigned to variable A at the beginning of the program. This is called variable initialization.

Line numbers from 1 to 9999 can be optionally used in the PB-700. However, consideration should be given when line numbers are allocated because program execution is performed in a sequence with the smallest line number first.

A program can be easily checked and modified by using line numbers based on 10.

Line 20 can also be expressed as follows.

20 LET A = 0 **See page 203.**

LET is an assignment command that can be omitted.

Now, press ↵.

[3] 30 INPUT "NUMBER" ; N **See page 197.**

Line number Input command Message statement Variable

Since INPUT is an input command statement, execution is not shifted to the next line unless the input of a numeral or character is performed by a key entry. A numeral or character that was entered is assigned to the variable after the message statement, and execution shifts to the next line.

Although a message statement can be omitted, the item for input can be displayed by inserting it.

When " ; " is placed after the message statement of INPUT, " ? " is displayed after the message. If " , " is placed before a variable instead of " ; " " ? " is not displayed.

Now, press ↵.

[4] 40 A=A + 1

Variable A is a counter that computes the number of program additions. Since 0 is assigned to A with the initialization on line 20, when line 40 is executed first, 1 is assigned to A based on A=0+1. When line 40 is executed again with "A = 1", 2 is assigned to A based on A=1+1.

1 is added to A depending on the number of executions on line 40 as mentioned above.

SEE

Line 40 execution:	1st time	A = 0 + 1	(A is 1)
	2nd time	A = 1 + 1	(A is 2)
	3rd time	A = 2 + 1	(A is 3)

Press the ↵ key.

[5] 50 B = N * A

This is an assignment statement the same as line 40 which is an instruction to assign the value of N * A to the numerical variable B. When this line is first executed, 1 has been assigned to A by the execution of line 40. An optional numerical value has been assigned to N by the execution of INPUT on line 30.
Therefore, if N is 17,

B = 17 * 1 ——→ 17 is assigned to B.

SEE In the following repeated execution on line 50, since 2 is assigned to A the second time,

2nd time B = 17 * 2 } The multiple of N is continuously assigned to the numerical variable B.
3rd time B = 17 * 3
⋮

Do you understand so far? The computations of line 40 and line 50 are repeated many times in this program. The repeat instruction will be explained later. If you understand the assignment statement, let's learn the following BASIC command. Press ⏎.

[6]
```
60   IF   B>200   THEN   100
```
Line number to which a jump is made.
Conditional expression
Decision statement

This is a conditional statement that says "If (IF) the value of B is larger than 200 (B>200), jump to line 100." In other words, when line 50 is repeatedly executed and the value of B becomes 204 after 12 times, B>200 is realized and a jump (branch) is made to line 100. If the value of B is equal to 200 or smaller, execution shifts to the next line without a jump.

IF expression THEN line number **See page 194.**

If this expression is realized, the program jumps to a specified line number.

This command was used. Now press ⏎.

[7]
```
70   PRINT   B ;
```
See page 205.

Numerical variable
Screen display command

PRINT is a display command. In this program, this command displays the content of numerical variable B on the screen. The semicolon after B is used to provide a continuous screen display. Because of this, the following display is performed without line change.

[8]
```
80   INPUT   " OK" ; C$
```
See page 197.

Character variable
Input command
Message

The INPUT statement you learned in the section on line 30 is used again on this line to perform character key input. When line 80 is executed, "OK?" is displayed as a message statement by which the key input of up to 7 characters is performed to be assigned to character variable C$. If numerical variable C is used here, a numerical value is only accepted as key input, and if a character or ⏎ is entered at this time, an SN error occurs.
The function of this line is to temporarily stop the display of the computation result of line 70 by using the function of the INPUT statement which waits for key input. If this line is not provided, many display results by program repeat are displayed at one time.
Press the ⏎ key.

[9]
```
90   GOTO   40
```
See page 192.

Jump command
Line number to which a jump is made

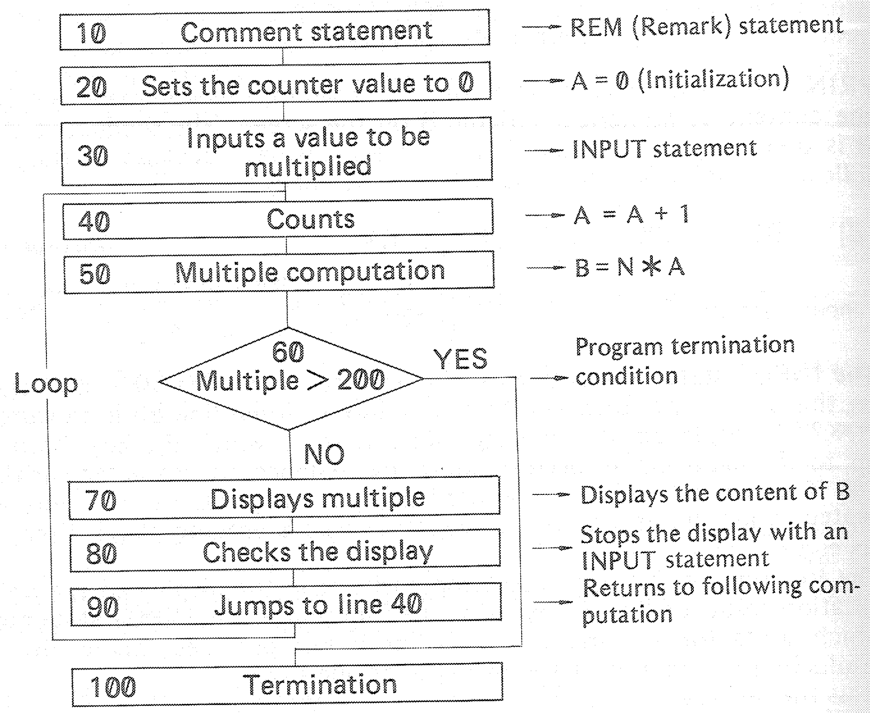
This is a command for unconditionally going to line 40. Press the ⏎ key.

[10]
```
100  END
```
See page 178.

END is a command that terminates a program. END is an essential command for a program because if a subroutine follows without an END command, the subroutine is also continuously executed. See page 188 for subroutines.
END can be inserted in line 60 as follows. If this is done, line 100 is not required.

```
60 IF B>200 THEN END
```

Let's observe the program flow as shown in the flow chart below.


10 Comment statement
20 Sets the counter value to 0
30 Inputs a value to be multiplied
40 Counts
50 Multiple computation
60 Multiple > 200
YES
NO
Loop
70 Displays multiple
80 Checks the display
90 Jumps to line 40
100 Termination
REM (Remark) statement
A = 0 (Initialization)
INPUT statement
A = A + 1
B = N ∗ A
Program termination condition
Displays the content of B
Stops the display with an INPUT statement
Returns to following computation

Since line 90 causes an unconditional return to line 40, this program repeats from line 40 to line 90. The program jumps out from a loop by the conditional statement on line 60 written in the middle of the loop, and jumps to line 100.

EXERCISE

- **Prepare a program for accumulation calculation by providing for the continuous input of numerical values.**

Hint

(1) Clear the variables to make them 0.
(2) Ask for the numerical value to be added.
(3) Add that numerical value to the previous numerical value.
(4) Display the result.

Answer

```
10  CLEAR
20  INPUT  "DATA =";A
30  B = B + A
40  PRINT  "TOTAL =";B
50  GOTO  20
```

Explanation

CLEAR on line 10 is a command that clears all numerical and character variables. In this case, only variable B for accumulation calculation may be cleared by B=0. If the GOTO command on line 50 causes a jump to line 10, the variable becomes 0 and accumulation cannot be performed.

3-8 PROGRAM EXECUTION

Now the program that was input has been checked. Did you operate the keys as requested? Do you understand how a BASIC program is prepared? Press the ↵ key.
Let's execute this program.
Press the BRK key and the CLS key first. Then enter the program execution command R U N ↵ or SHIFT $\overset{\text{RUN}}{\text{Q}}$ ↵ . Also, if a program is written in the P0 area, SHIFT $\overset{\text{P0}}{\text{0}}$ functions the same as RUN.
If program input has been correctly performed, the following is displayed.

```
RUN
NUMBER ?_
```

The input of the value to be multiplied is requested. Perform the following key operation.

17 ↵

Then the following is displayed.

```
RUN
NUMBER ? 17
17 OK ?_
```

The display shows that the minimum multiple of 17 is 17. Also, a confirmation is requested. Press the ↵ key to display the next multiple.

```
RUN
NUMBER ? 17
17 OK ?
34 OK ?_
```

Next, press the ↵ key for the following multiple. After this, multiples up to 187 are displayed by repeating this operation. If you press the ↵ key again and the next multiple does not appear, the conditional expression on line 60 has been realized. Since the multiple exceeds 200, program execution terminates. To execute the program again, press SHIFT $\overset{\text{RUN}}{\text{Q}}$ ↵ again.

Program Execution Sequence

```
NUMBER? 17
17  OK?
34  OK?
51  OK?
68  OK?
85  OK?
102  OK?
119  OK?
136  OK?
153  OK?
170  OK?
187  OK?
Ready  P0
```

Program Modification

To learn more about the use of various commands, further program corrections shall be made.
First, multiples up to 300 are obtained by changing B>200 on line 60 to B>300. Line 60 appears by SHIFT $\overset{\text{EDIT}}{\text{E}}$ 60 ↵. Move the cursor in order to change 200 to 300. Next, press the ↵ key and BRK key, then enter SHIFT $\overset{\text{RUN}}{\text{Q}}$ ↵ .
Let's check how the display is changed by changing the PRINT statement on line 70.
The content of line 70 is "PRINT B;". The semicolon after B functions to continue the display as you already know. Now, let's delete the semicolon.

SHIFT $\overset{\text{EDIT}}{\text{E}}$ 70 ↵

Display line 70 by the above operation, move the cursor to the position at " ; " and press the DEL key. Did the " ; " disappear? Press the ↵ and BRK keys and run the program. I'm sure that you know how to run the program already.
Message statement "OK?" is displayed under the multiple. In other words, since " ; " disappeared, the next character is not displayed continuously, but is displayed with a line change.

3-9 DISPLAY SCREEN CONFIGURATION

Now, let's learn the techniques of the screen display control by changing the multiple computation program on line 70 that was already entered.

BRK SHIFT EDIT E 70 ↵

Did the following display appear ?

70 PRINT B **See page 205.**

Let's display variable A which is used as the program repeat counter at the same time.

[1] 70 PRINT A ;B ;

Move the cursor to the position of B to make a modification, then enter SHIFT INS DEL SHIFT INS DEL . Since a 2 character space is provided before B, enter A SHIFT ; : ⇒ SHIFT ; : . Don't forget to press ↵ next. Execute the program to confirm the display.
Next, the following modification is performed.

[2] 70 PRINT A, B ;

To perform this modification, display line 70 in the EDIT mode after pressing BRK . Then make a correction by moving the cursor and press ↵, the same as above. After the modification is completed, execute the program.
The line change is performed by "A, B;".
Let's make another modification as follows. Press BRK .

[3] 70 PRINT A ; TAB (8) ; B;

To perform this modification, specify the EDIT mode as mentioned above, then move the cursor for correction and press ↵. Now, execute the program.
The TAB (8) function makes the spaces to the position which is specified by the number inside ( ). Confirm there is a space between the content display of variables A and B. Now, do you notice that a problem has occurred. In other words, when 3 numerals are required for the display of numerical variable B, the left side is arranged and the last character is shifted as shown below.

```
3         51  OK?
4         68  OK?
5         85  OK?
6         102  OK?
```

Since this is a display of a multiple, this is not too much trouble. However, when a quantity or price is displayed, the right side shall be arranged. To meet this requirement, it is rewritten as follows by using the USING function.

[4] 70 PRINT A; TAB (8); USING "###" ; B;

See page 267 for the use of USING. Now, execute the program to confirm the display. The LOCATE command also controls the screen display. Let's rewrite line 70 by using this command.

[5] 70 CLS : LOCATE 5,2 : PRINT A; B; . . .
Command Command Command

A line in which more than two commands are connected by " : " like this is called a multistatement.

Don't forget to press ↵ after modification. Is the display as shown below?

```
        1  17   OK?
```

The display is made at the center of the screen.
The LOCATE command is used as follows.

LOCATE X, Y

The number of horizontal characters is written in X while the number of vertical lines is written in Y. See page 204 for details.

3-10 REPEATING A SPECIFIED NUMBER OF TIMES

In a program, it is often necessary for some task to be repeatedly executed (called a routine) a certain specified number of times.
For example, this is used so that only required data is searched for from among considerable data, or so that considerable data is arranged in a sequence with the largest number first. A computer executes these tasks faithfully by checking that all data meets selection standards, by checking the size of adjoining data, and by sorting adjoining data. In this case, the commands you are going to learn are essential.
Let's start with a simple program. Check which program area is empty before inputting the program.

SHIFT SYSTEM S ↵ — See page 162.

The following display appears after this entry is made.

```
P    ♥♥23456789
 1470   Bytes :ANGLE 0
Ready  P0

_
```

Since the ♥ location has a program already, P0 and P1 are being used in the example display above. The remaining memory capacity is indicated as 1470 bytes. In this display, 1470 bytes can still be used freely. The remaining memory capacity is changed by the number of RAM expansion packs. (If nothing is stored, the memory capacity is 2864 bytes.)
ANGLE is used to specify the angle unit (see page 164) with a value from 0 to 2. Whatever this value is, it has no influence on ordinary computations. When the power is turned on, ANGLE is always set as 0 (Degree).
Ready P0 indicates the present program area which a program can be written in and executed. Next, move to an empty program area with PROG Number ↵ so that the following program can be entered.

```
10 CLS
20 FOR A=1 TO 20
30 PRINT CHR$(135);
40 NEXT A
```

Did you provide the correct input? Enter the following to confirm this.

SHIFT EDIT E ↵ ————→ Specifies the EDIT mode

If there is no mistake, display the following line with ↵.
After "Ready P0" is displayed, run this program.
This program displays ■ continuously 20 times.
Now, you will learn some new commands.

[1] 10 CLS

CLS is a command that clears the screen and moves the cursor to the upper left corner. It is used to make a display after the screen is cleared.

[2]

```
20 FOR A=1 TO 20
30 PRINT CHR$(135);
40 NEXT A
```

Line 20 and line 40 consist of one command. In other words.

FOR A = 1 TO 20 .... Assigns a numerical value from 1 to 20 sequentially to variable A.
NEXT A .... If A < 20, 1 is added to the value of A and a return is made to FOR.

This is called a FOR/NEXT loop. Let's follow the execution procedure.
(1) 1 is assigned to A.
(2) Executes line 30.
(3) Performs a comparison of A<20 by NEXT A on line 40.
This inequality is realized.
(4) Returns to line 20 and assign 2 to A.
(5) Executes line 30.
(6) Checks if A<20 is realized on line 40. It is realized.
(7) Returns to line 20, and assign 3 to A.

⋮

20 is assigned to A. Then A < 20 is not realized. As a result, the following line is executed. Since there is no program to follow in this case, "Ready P Number" is displayed.
Let's look at line 30 where the execution is repeated.

[3] 30 PRINT CHR$ (135) ; **See page 250.**

Display — CHR$ (135) — continuously

CHR$(135) indicates character dollar 135. Number codes from 0 to 255 are used to indicate all the characters and keys (characters) that can be used. (See the Table on page 321.)
Since a display of "■" cannot be entered directly by key input, it is fetched by specifing CHR$ ( ). Since this function is essential especially for characters that can not be entered directly by key input as mentioned above, you should become familiar with inserting numerals from 32 to 254 into CHR$ ( ).
"CHR$ (" is entered by SHIFT CHR$(A .

EXERCISE

■ **Prepare a program in which an integral from 1 to an optionally specified numeral is continuously displayed by using FOR/NEXT.**

Hint

```
FOR  A = 1   TO   N
          ∫
     NEXT  A
```

Answer

```
10  CLS
20  PRINT  "NUMBER"
30  INPUT  N
40  FOR  A = 1   TO  N
50  PRINT  A;
60  NEXT  A
70  END
```

Explanation

To provide for the input of an optional numeral on line 20, "NUMBER? " is displayed as a message statement. The entered numerical value is assigned to numerical variable N and the number of repeat times is specified on line 40. For example, if a numerical value such as 15 is entered to N, line 50 is executed the specified number of times.
The variable used by FOR/NEXT is displayed on line 50 as it is. From this, it is clear that 1 is added to the variable used by the FOR/NEXT command each time the loop is repeated.

3-11 SUM TOTAL PROGRAMMING

The FOR/NEXT command is hard to understand unless you become used to it. Let's prepare another program to become familiar with this command.
Let's determine the program area for inputting the following program. It is assumed to be P4. It is not necessary to explain the input procedure again. Now, input the following program after NEW ⏎.
In this program, an input of a determined number of articles is performed first, then the subtotal and total are displayed by repeating input of the unit prices and quantities.

```
 10 CLEAR
 20 INPUT "NUMBER OF ARTICLES";N
 30 FOR A=1 TO N
 40 INPUT "UNIT PRICE";B
 50 INPUT "QUANTITY";C
 60 PRINT "SUBTOTAL";B*C
 70 D=D+B*C
 80 NEXT A
 90 PRINT "TOTAL";TAB(10);"$";D
100 END
```

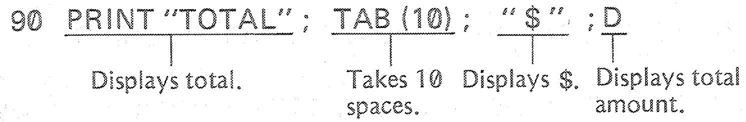
This program processes the task on each line as follows.

10 Clears all variables (assigns 0 to all variables).
20 Requests the input of the number of articles processed (Enters the number of articles to N).
30
Requests the number of articles (N), the unit price and quantity. After their input, the subtotal is diaplayed and it is added to the total.
80
90 Displays the total amount.
100 Termination

Let's analyze the FOR/NEXT loop from line 30 to line 80 in detail. The task to be performed in the loop from FOR to NEXT is as follows.

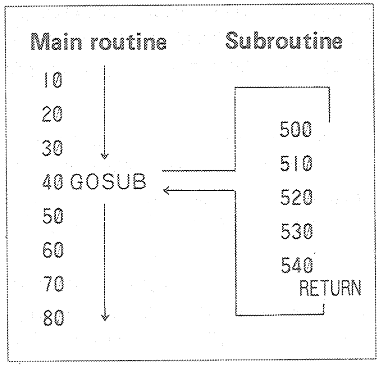
(1) The unit price input is made and it is assigned to variable B.
(2) The quantity input is made and it is assigned to variable C.
(3) A computation of unit price x quantity is performed and the subtotal is displayed.
(4) The subtotal is added to the total to provide a new total.

On line 90 of this program, an easy display is considered.



```
90 PRINT "TOTAL"; TAB (10); "$"; D
```

Let's rearrange the program based on a subroutine concept. Separate the task into some blocks in the FOR/NEXT loop so that when the program advances to that task, it jumps to that block, and it returns to the original location after completing the task. At the same time, an easy display is contrived.
The concept of this subroutine is as shown on the right. The command used for this is GOSUB/RETURN.
Lines 40 to 70 in the previous program are changed to lines 500 to 540 according to the figure on the right, and GOSUB 500 is inserted in line 40.



GOSUB 500

Go to the subroutine on line 500.

RETURN

Return to the command next to GOSUB.

Lines 30 to 80 are modified as follows by the above.

```
30 FOR A = 1  TO  N
40 GOSUB  500
50 NEXT  A
```

This has been arranged to be placed in one program.

```
 10 CLEAR
 20 INPUT "NUMBER OF ARTICLES";N
 30 FOR A=1 TO N ─┐
 40 GOSUB 500     │ FOR/NEXT loop. Repeats N times.
 50 NEXT A       ─┘
 60 PRINT "TOTAL";TAB(10);"$";D
 70 END
500 PRINT A;TAB(5);"UNIT PRICE";:INPUT─┐
B                                      │
510 PRINT A;TAB(5);"QUANTITY";:INPUT C │ Subroutine
520 PRINT "SUBTOTAL";B*C:BEEP 1        │
530 D=D+B*C                           ─┘
540 RETURN
```

Since this program has many modifications compared with the previous program that was rearranged by using the EDIT mode, input this program in a new program area. Confirm the execution differences with those of the previous program.

3-12 CHARACTER VARIABLES

A method for storing lots of data, explained in "3-10 Repeating A Specified Number of Times", will be learned as follows. However, since it is necessary to become used to handling character data, let's practice using character variables.

Variables are roughly divided into the following two categories as you already know.

Only numerical values can be assigned ....
Numerical variables (such as A, B, C, A1, etc.)

Characters and symbols can be assigned ....
Character variables (such as A$, B$, A1$, etc.)

What is the difference between a numerical variable and a character variable in a program? The "Characters" used in character variables contain alphabetical characters A, B, C ....., Kana, symbols and numerals (0—9). Therefore, the difference between numerical values and characters may not be clear.

However, the decisive difference between these two variables is that numerical values express a size. Therefore, a numerical value used in a character variable is the same as a symbol, and does not express a size.

Now, let's check the difference.

[Used as a numerical value]	**[Used as a character]**
4 + 3 ENTER → 7	"4" + "3" ENTER → 43

A character must be placed inside " ", the same as message statements for INPUT and PRINT commands. In the above example, characters 4 and 3 are displayed by linking them. A numeral treated as a character variable can be converted to a numerical value with the VAL function (see page 252).

Character variables are often used in array variables which are explained in the following section. In this case, up to 16 characters can be entered to a variable unless this is specially specified (declared). This is called fixed-length character arrays.

However, this is sometimes not so convenient. Therefore, when many characters are to be entered such as for an address book, up to 79 characters can be entered to a variable.

In this case, since a large memory capacity is assumed, only minimal data can be entered (see page 27). The declaration method is as follows (called a defined-length character array).

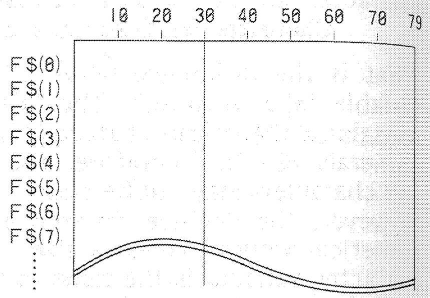
```
DIM  F$ (50) * 30
```

Character variable to be used

Number of data

Maximum characters per variable

Memory can be saved as shown in the right figure which allows the number of data items entered to be increased.
When 30 is declared, the memory required in the above example is as follows.

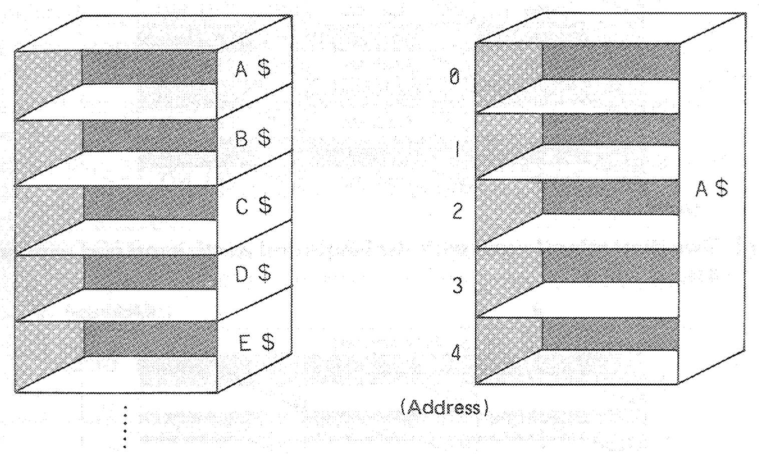


$$50 \times \underbrace{(30+1)}_{\text{Number of characters stored + 1}} = 1550 \text{ (Bytes)}$$

To enter as much data as possible, the memory used per variable is set up to be reduced as little as possible in an array program.
In the case of a numerical variable, for example, to meet this requirement, a half-precision or single-precision numerical array can be selected. A half-precision numerical array requires only half of the memory used for a single-precision numerical array which will be explained in detail in the following section.

3-13 WHAT IS A DIMENSION?

Have you ever heard a programmer say "Take a dimension"? This means "prepare a container". You can imagine storage shelves as shown in (2) of the figure below.
Data (numerical value and character-string to be stored) can be placed on each shelf.
In this example, 5 shelfs are prepared for one variable name. Since many different names are provided for one variable such as "SMITH" in A$ No. 0, "JOHNSON" in A$ No. 1, and "FOSTER" in A$ No. 2, etc., a required name can be called by using a Control No.
If data is not stored in a variable by number placement, only one data item can be placed in one variable as shown in (1). Therefore, numerous variable names are required, while variable data cannot be controlled during preparing a program, which is very inefficient.



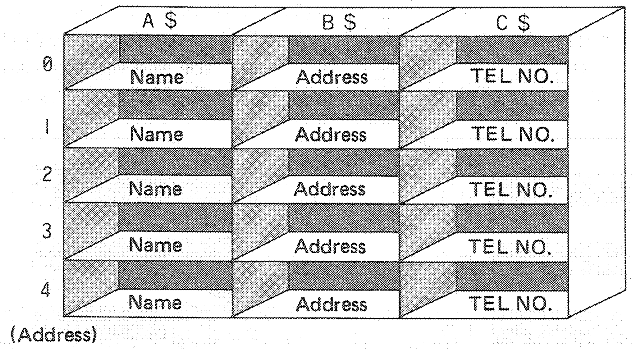
(1) 1 data item for one variable — inefficient

(2) Many data items are entered for one variable name — one dimensional array.

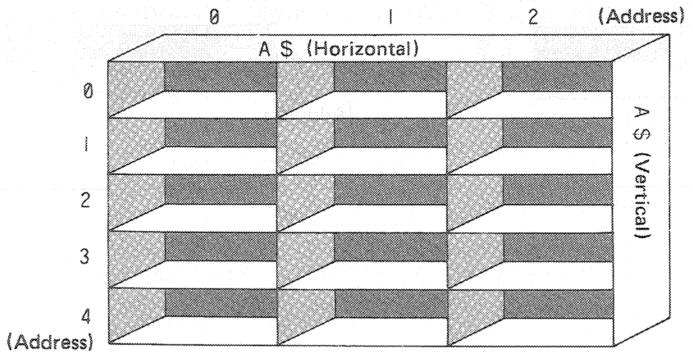
■ One-dimensional Arrays

To enter many data items, the method of (2) is used. This is called a one-dimensional array which means that data are arrayed in one variable name by undirectionally placing numbers. In the figure, although the numbers used are 0—4, up to 256 shelves from address 0 to address 255 can be actually set up.
When such a data array is used, it is necessary to reserve memory such as the maximum number of storage shelves in the variable name A$. Otherwise, when data is entered, the computer cannot determine the space (memory) where the data is entered, or in other words, the computer cannot judge how many storage shelves should be prepared in the variable name A$, and an error occurs.
The length and capacity of the storage shelves of A$ is called a dimension of A$ (size).

(3) Address book in which some one dimensional arrays are arrayed.



(4) Two-dimensional array with the length and width expressed with one variable name.



■ Two-dimensional Arrays

Let's change the form of the storage shelves.
A list of names and an address book can be prepared by using a one-dimensional array as shown in (3) by placing names on the array shelf A$, addresses on the array shelf B$, and telephone numbers on the array shelf C$ as an example. In a program, all you have to do is enter a name, then you can call B$ and C$ on the address of this name.
To provide a method to control each shelf, addresses are placed vertically and horizontally, and the shelf location can be specified by a horizontal and vertical address combination.
This viewpoint can be expressed as shown in (4), a two-dimensional array, which means that data is arranged in one variable name by placing numbers bidirectionally (vertically and horizontally).
In this case, addresses from 0 to 255 can be placed both vertically and horizontally. However, since the memory capacity has a limitation, they cannot be placed up to 255 both vertically and horizontally at the same time.
The memory required for a two-dimensional array is: Number of vertical addresses x Number of horizontal addresses x Number of required memories per address.
Therefore, if 256 addresses are placed both vertically and horizontally for a fixed-length character variable, in which up to 16 characters can be entered per variable;

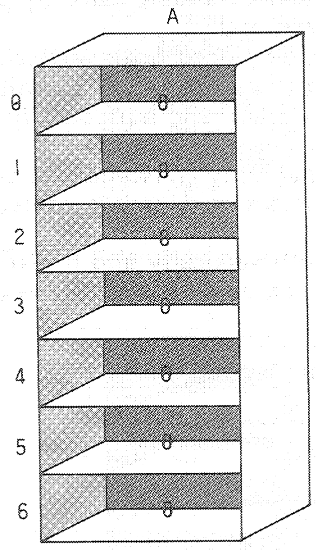
$$256 \times 256 \times 17 = 1114112 \text{ bytes.}$$

As a result, memory overflow occurs.

3-14 NUMERICAL ARRAY VARIABLES

A numerical array variable indicates variable utilization when a numerical value is placed on a storage shelf (array) that was analyzed in the previous section. An array that arranges many data items in one variable name is called an array variable. In the storage shelf example of the previous section, A$, B$, and C$ are the array variables.
Although the utilization of an array variable to control numerical values and character strings is basically the same, data handling and expression are different in a program.
First we will learn how to prepare a program using numerical variables. Let's go back to the one-dimensional storage shelf as shown in (1). Now let's name the storage shelf. This name must be one alphabetical character, one of 26 fixed variables. Let's call it "A" here.
In this case, the storage shelf of array variable A consists of 7 shelves from 0 to 6.

(1) A Configuration and expression of array variable A



DIM A(6) is written for this storage shelf.

Precaution!
Even if there are 7 data items, since 7 is from 0 to 6, the maximum value is 6.

In the computer, first prepare storage shelf A which has shelves from 0 to 6. In other words, a declaration must be made that A is used as an array variable and its size (dimension) is 0 – 6.
The method used is as follows:

Numerical array name

DIM A (6) ⏎

Command to take the dimension on the right

7 shelves from 0 to 6 with a one-dimensional array. Placed in ( ) to show it is an array.

When the above is specified, all arrays of the numerical array variable A become 0.
If a DD error occurs when you perform this operation, perform the above operation again after entering CLEAR ⏎.
Let's place many different data items on the shelf of (1).

A (5) = – 13 ⏎

Address 5 in array variable A

A (3) = 65 ⏎

Let's confirm that data was placed on shelves A(5) and A(3) by this operation.

A (5) ENTER ➡ Displayed as –13

A (3) ENTER ➡ Displayed as 65

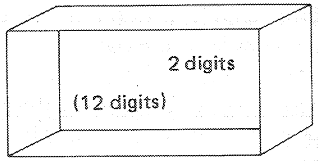
These operations can be written in a program as follows.

```
 5 CLS
10 CLEAR
20 DIM  A(6)
30 INPUT  "A(5) = "; A(5)
40 INPUT  "A(3) = "; A(3)
50 PRINT  A(5); A(3)
60 END
```

■ Knowledge Concerning A Numerical Value Used For An Array Variable

It is certain that the method in which numerical data is stored on an array storage shelf has been understood based on the explanation mentioned before. What restrictions are there on numerical values that can be entered in array variables? The important thing is that a 12 digit integer and a 2 digit exponent can be entered in one array variable. However, it should be noted that it is displayed on the screen with 10 digits by rounding the 11th position. The computation is performed with 12 digits and the display is made with 10 digits. The numerical value used in a numerical array variable like this is called single precision. On the other hand, often 12 significant digits are not necessary, and 10 digits are not necessary for the display since 5 digits are sufficient. In this case, there is a very convenient method for storing numerical values which shall be mastered. This is a processing method of numerical value, called a half-precision, by which a numerical value is stored in an array variable with up to 5 digits. The required memory for one data item is one-half of that for single-precision (4 bytes) when this method is used.

(5) Numerical values used in a numerical array.



Since the 12th digit and after are rounded, they are not significant digits.

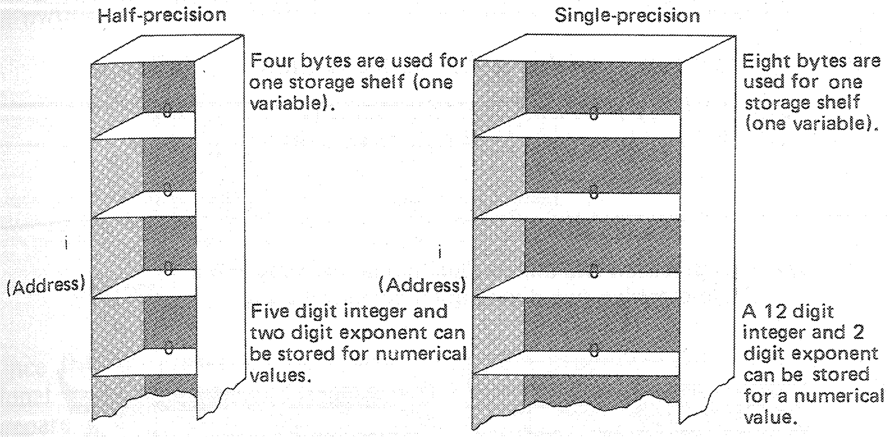
(Display of Number of Digits)

In the PB-700, numerical values can be expressed with up to 100 digits. Also, more digits can be displayed depending on the program capability. However, if 10 digits are exceeded, an exponential display is performed on the screen as follows, for example.

$$12345678909\ \boxed{\text{ENTER}} \Rightarrow \underbrace{1.23456789\underbrace{1}_{\text{Rounded}}}_{\text{Integer}}\ \text{E}\ \underbrace{10}_{\text{Exponent}}$$

The exponential display is an expression of "x 10^{10}" which is 1.234567891 x 10^{10} in the above example.

(6) Difference between half-precision and single-precision is the shelf width sizes.



Four bytes are used for one storage shelf (one variable).

Five digit integer and two digit exponent can be stored for numerical values.

Eight bytes are used for one storage shelf (one variable).

A 12 digit integer and 2 digit exponent can be stored for a numerical value.

The difference between single-precision and half-precision is as shown in (6).

When the variable is A, a single-precision dimension is expressed as shown below, as previously mentioned.

A(*i*) ......Declaration of a single-precision numerical array.

However, this half-precision dimension is expressed as shown below in which ! is required after the variable.

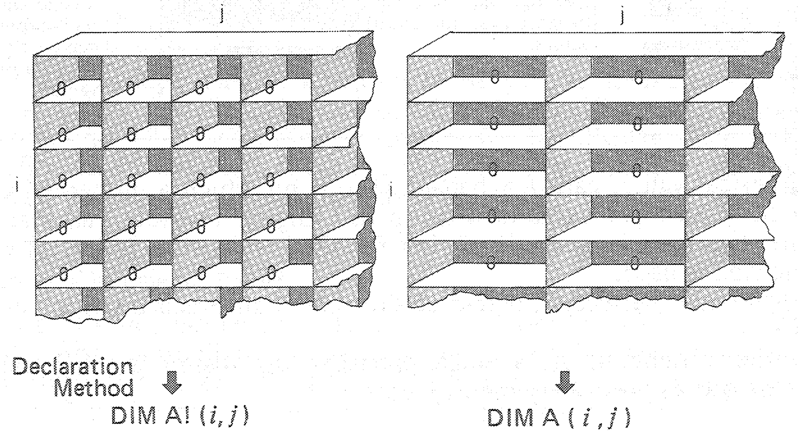
A! (*i*) .... Declaration of a half-precision numerical array.

■ Viewpoints of half-precision and single-precision in a two-dimensional array

The viewpoint is exactly the same as that for the previously mentioned one-dimensional array. An example of the storage shelf is as shown in (7).
If the numerical array variable is A, the declaration is as follows.

DIM A! (*i* , *j*) .... Half-precision.
DIM A (*i* , *j*) .... Single-precision.

(7) **Half-precision two-dimensional numerical array and a single-precision two-dimensional numerical array.**

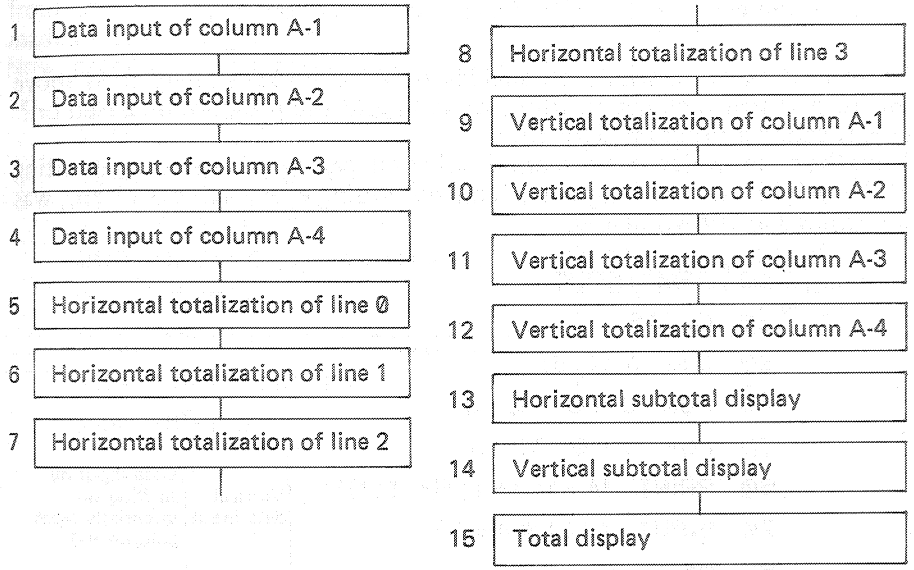

j
j
i
i
0
Declaration Method
DIM A! (*i*, *j*)
DIM A (*i* , *j*)

3-15 NUMERICAL ARRAY PROGRAMMING

Let's prepare a program for a vertical and horizontal totalization table. Vertical and horizontal subtotals are to be computed, and a total is obtained at the end. The vertical and horizontal data is 4 and 4, respectively, and 16 in total.

	A-1	A-2	A-3	A-4	Subtotal
0	25	17	3	67	
1	19	20	11	58	
2	32	15	26	55	
3	36	28	29	40	
Subtotal					

Since this is numerical data, let's prepare a program with a one-dimensional numerical array. Let's select A as the variable name. Now, let's prepare the program with the following procedure which is called a flowchart.


1 Data input of column A-1
2 Data input of column A-2
3 Data input of column A-3
4 Data input of column A-4
5 Horizontal totalization of line 0
6 Horizontal totalization of line 1
7 Horizontal totalization of line 2
8 Horizontal totalization of line 3
9 Vertical totalization of column A-1
10 Vertical totalization of column A-2
11 Vertical totalization of column A-3
12 Vertical totalization of column A-4
13 Horizontal subtotal display
14 Vertical subtotal display
15 Total display

Let's prepare a program for data input to each column explained on the flow chart 1 to 4.

```
10 CLEAR
20 DIM  A!(15)
30 FOR  I=0  TO  3
40 INPUT  A!(I)
50 NEXT  I
```

FOR-NEXT loop

With this program, data can be entered to lines 0–3 of column A-1, however, how do you enter data to column A-2? This is a problem because a method, in which input is only performed with the array A without using another variable, has to be found.

	→ I				Subtotal
	A!(0)	A!(4)	A!(8)	A!(12)	B!(0)
↓	A!(1)	A!(5)	A!(9)	A!(13)	B!(1)
J	A!(2)	A!(6)	A!(10)	A!(14)	B!(2)
	A!(3)	A!(7)	A!(11)	A!(15)	B!(3)
Subtotal	C!(0)	C!(1)	C!(2)	C!(3)	D

To enter data sequentially on shelves A!(0)–A!(15) as mentioned above and to perform each totalization, what should the program be based on? This is called a program algorithm.

A method, in which a horizontal subtotal can be obtained by treating A!(0)–A!(15) as one long shelf, and by adding every 4th data item, was determined as shown below.

```
10 REM INPUT
20 CLEAR
30 DIM  A!(15)
40 FOR  J=0  TO  3
50 FOR  I=0  TO  3
60 PRINT  "A-";J+1;"(";I;")";
70 INPUT  A!(J*4+I)
80 NEXT  I
90 NEXT  J
```

Vertical data input

Data input by shifting sequentially from column A-1.

This completes the data input program. Data is sequentially entered with the display of "A–Column No. (Vertical No.)?" Data is stored in array variable A! ( ) on line 70.

Now, let's prepare a program for the horizontal subtotal!

```
 95 DIM B!(3)
100 FOR  I=0  TO 3
110 FOR  J=0  TO 3
120 B!(I)=B!(I)+A!(J*4+I)
130 NEXT J
140 PRINT "B-";I;TAB(5);B!(I)
150 INPUT "OK";F$
160 NEXT  I
```

Horizontal computation and display.

Computes the subtotal by shifting vertically 4 times and displays it.

Data is stored to array variables B!(0)–B!(3) by line 120, and is displayed by line 140. To prevent display of the subtotal of B!(0)–B!(3) from scrolling, the display stops by line 150, and the next display is made by entering ⏎.

Now, prepare a program for the vertical subtotal.

```
170 DIM C!(3)
180 FOR  J=0  TO 3
190 FOR  I=0  TO 3
200 C!(J)=C!(J)+A!(J*4+I)
210 NEXT I
220 PRINT "C-";J;TAB(5);C!(J)
230 INPUT "OK";F$
240 NEXT J
```

Vertical computation and display

Computes the subtotal by shifting horizontally four times and displays it.

This is almost the same as the horizontal subtotal method using a double FOR-NEXT loop with a different sequence of variables (loop control variables) I and J.
Now, let's compute the total.

```
250 REM TOTAL
260 FOR  I =0 TO 3
270 D = D+C!(I)
280 NEXT  I
290 PRINT "TOTAL=";D
300 END
```

Programming With A Two-Dimensional Array

The handling of each shelf where data is to be stored is complicated and difficult to understand when a one-dimensional array is used as previously explained. Let's prepare a program using the two-dimensional array you have learned, since it is very convenient for handling vertical and horizontal data.

1 Initialization

```
10 ERASE A        .... Clears the array of variable A.
20 CLS            .... Clears the screen.
30 N=4            .... The number of vertical or horizontal items.
40 DIM A!(N,N)    .... Dimension declaration (The declaration of half-
                       precision numerical array).
```

In regard to line 30, N = 4, the number of vertical and horizontal items can be changed by changing this numeral.

2 Data Input

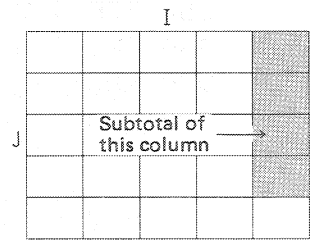
```
 50 FOR I=0 TO N-1
 60 FOR J=0 TO N-1
 70 PRINT I;"-";J;
 80 INPUT A!(I,J)
 90 NEXT J
100 NEXT I
```

J ↓	→ I				Horizontal total
	28	39	12	54	133
	53	29	55	30	167
	28	17	80	53	178
	60	31	70	44	205
Vertical total	169	116	217	181	683

In regard to N − 1 of FOR-NEXT statements on lines 50 and 60, since a total column is not required for data input, it is N − 1. This time, data is entered in a horizontal sequence which is different from the previous one-dimensional array.

3 Horizontal Subtotal

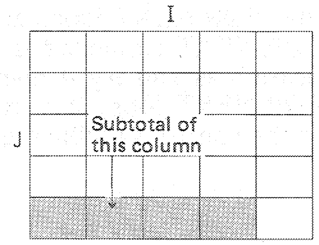
```
110 FOR I=0 TO N-1
120 FOR J=0 TO N-1
130 A!(I,N)=A!(I,N)+A!(I,J)
140 NEXT J
150 NEXT I
```



This performs a horizontal subtotal 4 times (when N = 4). Confirm the value of A! (   ) by entering 0 to N − 1 for I and J.

4 Vertical Subtotal

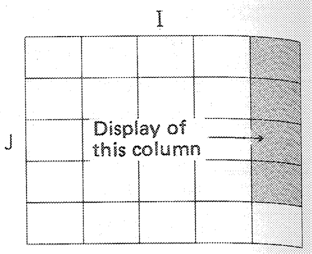
```
160 FOR J=0 TO N
170 FOR I=0 TO N-1
180 A!(N,J)=A!(N,J)+A!(I,J)
190 NEXT I
200 NEXT J
```



This performs a vertical subtotal 4 times. Confirm the value of A! (   ) by entering 0 to N − 1 for I and J.

5 Horizontal Subtotal Display

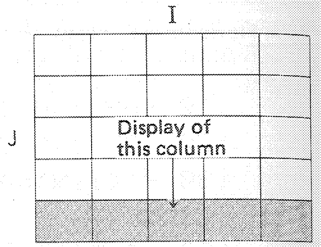
```
210 FOR I=0 TO N-1
220 PRINT A!(I,N);
230 NEXT I
240 STOP
```



The program stops once and confirms each horizontal subtotal. Then, execution shifts to the next line by entering SHIFT CONT W ↵.

6 Vertical Subtotal Display

```
250 FOR J=0 TO N
260 PRINT A!(N,J);
270 NEXT J
280 END
```



This displays the vertical subtotals. Since totalling horizontal subtotals is performed at the same time, the total is displayed at the same time.

The difficult part of the vertical and horizontal totalization program is that FOR-NEXT loops are used doubly. However, you will find that the FOR-NEXT command is very convenient and useful for program simplification as you try it many times and become used to it. In regard to a FOR-NEXT loop, it is recommended that numerical values be entered to the control variable sequentially from 0.

7 Program Execution

When you run the program, "0—0? " is displayed which requests data entry. Enter Data ↵, then a data entry for the next horizontal column is requested by "0—1? ".
After a little while when all data input has been performed, the horizontal subtotals and "STOP P0—240" are displayed and the program stops.
Next, the vertical subtotals and total are displayed by entering SHIFT CONT W ↵ and the program is terminated. Try several different display formats by performing a rearrangement.

- **Control variables**

For example, in regard to the horizontal subtotal mentioned above, control variables change as follows.

```
110 FOR I=0 TO N-1
120 FOR J=0 TO N-1
130 A! (I,N)=A!(I,N)+A!(I,J)
```

Control variables

I and J are control variables of the FOR-NEXT loops. They change from 0 to 3.

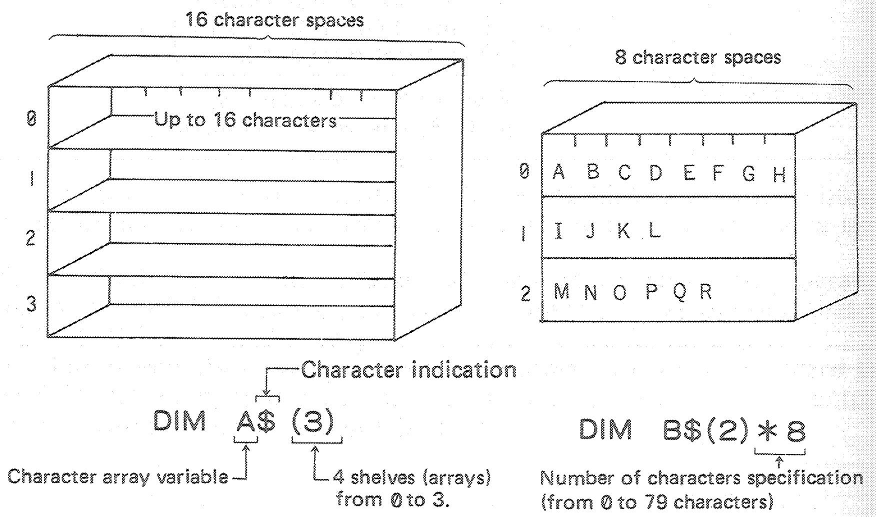
When I = 0, J = 0 and N = 4,
A! (0, 4) = A! (0, 4) + A! (0, 0).

3-16 CHARACTER ARRAY VARIABLES

One-dimensional and two-dimensional arrays can both be used for character array variables into which characters can be entered, the same as for numerical array variables. Also, half-precision and single-precision can be used in a character array variable which is useful for effective memory utilization, the same as for a numerical array variable. This is the proper use of fixed-length character arrays and defined-length character arrays.
It is wasteful to use a space, in which more than 5 characters can be entered, for an array variable that only requires space for 5 characters. On the other hand, if the space for only 10 characters is used while space for 20 characters is required, trouble occurs when data is entered. In other words, it is important to use a defined-length character array properly for using character array variables.
The number or characters per character array variable is specified as follows.

```
DIM A$(i) * n   (1 ≦n<79)
```

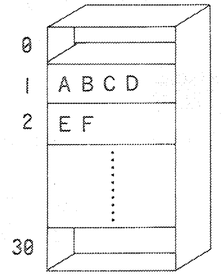
* If *n is omitted, the number of characters is specified to 16 characters.


16 character spaces
Up to 16 characters
0
1
2
3
8 character spaces
0
A B C D E F G H
1
I J K L
2
M N O P Q R
Character indication
DIM A$ (3)
Character array variable
4 shelves (arrays) from 0 to 3.
DIM B$(2)*8
Number of characters specification (from 0 to 79 characters)

* Since * n is omitted, the number of characters is specified to 16 characters.

Let's enter characters to an array variable. Perform the following operations.

```
CLEAR ⏎
DIM  H$(30)*5  ⏎
H$(1) = "ABCD"  ⏎
H$(2) = "EF"  ⏎
```


0
1
A B C D
2
E F
30

When you execute these entries, the characters are stored in the array variable as shown in the figure on the right.
To recall these, perform the following operations.

```
H$(1)  ENTER  ➡  ABCD
H$(2)  ENTER  ➡  EF
```

■ **Error in DIM Statement**

When more characters than specified by a DIM statement are entered,

```
ST  error
```

appears. If this happens, reenter the number of characters within the range.
Also, a DD error sometimes appears when a specification has been performed with the same variable name as that for the array to be specified (without any relationship to turning the power off). For example, if

```
DIM A$(20)  ⏎
```

was previously executed, and if

```
DIM A$(30)  ⏎
```

is now executed, an error will occur. When this happens, erase A$ by using the ERASE command as follows.

```
ERASE A$  ⏎
```

However, when ERASE or CLEAR is executed, precautions shall be taken to ensure that data is erased.

■ Character Array Programming

Exercise

Problem

Prepare an array program with 3 arrays in which up to 10 characters can be stored respectively. Characters are to be entered by character codes. To stop input in the middle, 0 is to be entered. When the entry for 3 arrays has been completed, all characters are displayed.

Hint

Since initialization is with up to 10 characters and 3 arrays,

```
10 CLEAR
20 DIM  N$(2)*10
```

The data input routine comes next. Since there are three arrays, prepare three FOR-NEXT loops, and use another loop specified by a GOTO statement to read 10 characters. Include an INPUT statement in the loop to read character codes.
The basic configuration of the data input program is as follows.

(1)	Sets the counter to 0 for the number of characters per variable	B = 0
(2)	Inputs a character code	INPUT N
(3)	Adds 1 to the counter for the number of characters	B = B + 1
(4)	If the counter exceeds 10, inputs character codes to next array.	IF B = 10 THEN ~
(5)	Converts a code number to a character and stores it to an array variable	N$(I) = N$(I) + CHR$(N)
(6)	Returns to (2)	GOTO ~

The routine for displaying the result comes next. Provide a continuous display of N$(0) – N$(2).

```
100 FOR  I=0  TO  2
110 PRINT  N$(I);"␣";
120 NEXT  I
```

········0 to 2 is sequentially entered to I.

Answer

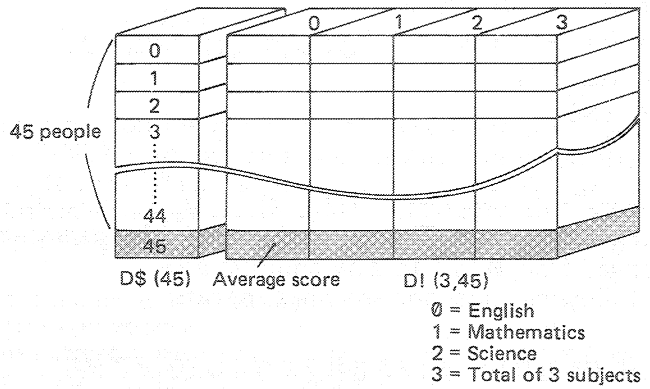
```
 10 CLEAR :CLS
 20 DIM N$(2)*10
 30 FOR I=0 TO 2:B=0
 40 PRINT "N$(";I;")";:INPUT "  No.";N
 45 IF N>255 THEN 40
 50 IF N=0 THEN 90
 60 B=B+1:IF B=10 THEN BEEP :GOTO 90
 70 N$(I)=N$(I)+CHR$(N)
 80 GOTO 40
 90 NEXT I
100 FOR I=0 TO 2
110 PRINT N$(I);" ";
120 NEXT I
130 END
```

When you execute this program, "N$(␣0)␣␣No.?_" is displayed. Now enter the character code. The numeral enclosed by parentheses of "N$(I):" is changed from 0 to 1 to 2 by entering 0 ⏎.
Now, enter the following codes and see what appears.

67 ⏎ 65 ⏎ 83 ⏎ 73 ⏎ 79 ⏎ 0 ⏎ 67 ⏎
79 ⏎ 77 ⏎ 80 ⏎ 85 ⏎ 84 ⏎ 69 ⏎
82 ⏎ 0 ⏎ 80 ⏎ 66 ⏎ 55 ⏎ 48 ⏎ 48 ⏎ 0 ⏎

3-17 COMBINATION OF CHARACTER ARRAYS AND NUMERICAL ARRAYS

In most cases when data is processed by preparing a vertical and horizontal table, a combination of character and numerical arrays is used. In this case, characters and numerical values must be handled at the same time as one data group.
For example, in regard to a name and score, or the names of articles, number of articles and an amount, characters and numerical values must be recalled at the same time. Let's understand how a program like this is prepared.
Let's prepare a result processing program as an easy example. First, a name is required. Then a two-dimensional numerical array is required in which the total score for three subjects (English, mathematics, and science) corresponding to a name is stored.
The following model can be assumed based on the items mentioned above.



The data to be entered is as follows.

Name	English	Mathematics	Science	Total score
A. Y.	50	60	75	
K. K.	83	71	70	
S. O.	60	63	40	
Average score				

1 Name Input

```
10 ERASE D$,D!:CLS
20 DIM D$(45)*10,D!(3,45)
30 I=0
40 INPUT "NAME ";D$(I)
50 IF D$(I)="END" THEN 80
60 I=I+1:IF I=46 THEN 80
70 GOTO 40
```

Names are determined by D$(0) to D$(44) and the title (in this case, AVERAGE SCORE) is determined by D$(45). Jump out from a loop is made by inputting "END". Of course, lines 40—70 provide the loop for name input.

2 Numerical Value Input

This is a routine in which numerical data is entered by a two-dimensional array. First, the name (D$(Y)) is displayed by line 90, and the English score for the name (when X is 0) is fetched to D! (0,0) by line 120. Then the score for the name is added to D! (3,0) by line 130. D! (3,Y) is a subtotal column. However if the horizontal subjects are to be increased, 3 can be replaced with the value in which 1 is added to that numeral. Also, X,Y of D! (X,Y) are determined by the frequency of the FOR-NEXT loop.

```
→FOR Y=0 TO I−1
  →FOR X=0 TO 2
      INPUT D!(X, Y)
   NEXT X
 NEXT Y
```

Its form is basically as shown below.

```
 80 FOR Y=0 TO I-1
 90 PRINT D$(Y)
100 FOR X=0 TO 2
110 PRINT X
120 INPUT " ";D!(X,Y)
130 D!(3,Y)=D!(3,Y)+D!(X,Y)
140 NEXT X
150 NEXT Y
```

Score table

→ X

↓ Y

English (0)	Mathe-matics (1)	Science (2)	Total

3 Subject Average Computation

Since D! (X,45) was specified as the column of subject average score without any relationship to the name input frequency (Counter I), the average score for each subject is entered here. First, the total score of each subject is assigned to D! (X,45) on line 180, the average score is assigned to D on line 200, and the second decimal place of D is rounded up on line 210 with values below the decimal point discarded for integers, and the value is assigned to D! (X,45) again. In other words, only when the first decimal place of average score is 9, it is rounded up and otherwise discarded. INT is the integer function in which values below the decimal point are discarded.
This occurs because of characteristics of test result processing. If used for other purposes, line 210 is changed.

```
160 FOR X=0 TO 3
170 FOR Y=0 TO I-1
180 D!(X,45)=D!(X,45)+D!(X,Y)
190 NEXT Y
200 D=D!(X,45)/I
210 D!(X,45)=INT(D*10+0.5)/10
220 NEXT X
```

4 Total Score Display For Each Name

The total score for each name is displayed. After you press ↵ on line 260, the program shifts to the next name. The total score for a name is processed by loop control variable Y. Therefore, the following always matches.

Name .... D$(Y)
Total score .... D! (3,Y)

```
230 FOR Y=0 TO I-1
240 PRINT D$(Y);" T=";
250 PRINT D!(3,Y)
260 K$=INKEY$:IF K$="" THEN 260
270 NEXT Y
```

5 Average Score Display

The display of the average score for each subject is performed by sequentially displaying data in case of X=0 to X=3 with D! (X,45).

```
280 FOR X=0 TO 3
290 PRINT "AVE.=";
300 PRINT D!(X,45)
310 K$=INKEY$:IF K$="" THEN 310
320 NEXT X
330 END
```

Execute this program and make data entries.

The display screen of execution result is as follows.

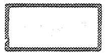
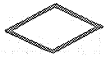
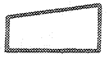
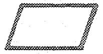
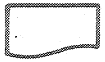
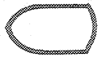
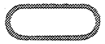
```
  A.Y T= 185 }
  K.K T= 224 } Total score for a name is displayed sequentially each
  S.O T= 163 } time ↵ key is pressed.
↵ AVE. = 64.3  ——Average score of English(0)
↵ AVE. = 64.7  ——Average score of Mathematics(1)
↵ AVE. = 61.7  ——Average score of Science (2)
↵ AVE. = 190.7——Average of each person's total score
```

3-18 HOW TO MAKE A FLOWCHART

When a program is being prepared, a flowchart is very convenient, and is useful even after a program has been prepared. After you have learned the BASIC commands and have mastered the preparation of a short program, it is recommended that you always prepare a flowchart.
The processing range and sequence of a long program can be clarified by making a flowchart which eliminates careless mistakes, excessive processing, and incorrect processing. Also, a flowchart is very useful when a program is rechecked or modified after a long time has elapsed.

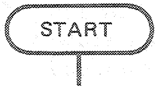
■ Flowchart Description Rules

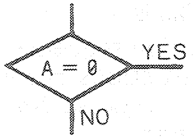
Although there are various kinds of flowchart symbols, the nine symbols shown below are sufficient.

Symbol	Meaning
	Process — Indicates all kinds of processing.
	Decision — Indicates a decision to select a path from among many.
	Manual input — Indicates input from keyboard.
	Input/Output — Indicates a data input/output.
	Document — Outputs to a printer or plotter.
	Display — Displays on a screen.
	Connector — Indicates an exit to another place or entrance from another place.
	Terminal — Indicates a flowchart terminal such as start and end.
	Flow line — Indicates connections of symbols mentioned above.

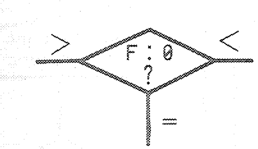
■ Using Symbols

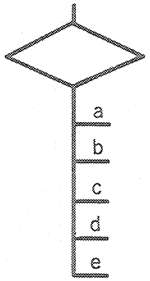
The direction of flow is basically from left to right and from top to bottom. When the flow is other than this, it is indicated by an arrow. Arrows shall be used as much as possible since they are easily understood.
Also, flow lines can cross.
A statement written with a symbol such as "Start" or "A = 0" shall be placed inside the symbol as much as possible.

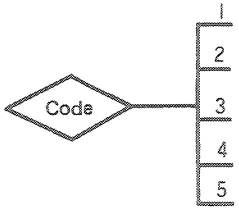




Although two outlets or more can be written (jumping) for one symbol, the jumping condition must be written at each outlet.





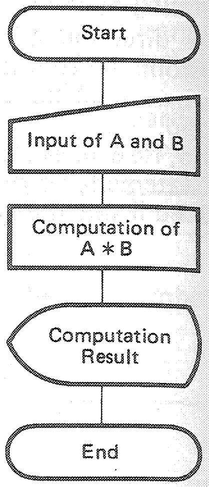


■ Flowchart Writing Procedure

The procedure for the task to be performed is placed in a flowchart. However, since it actually cannot be skillfully written, the flow viewpoint is explained as follows.
There are two kinds of flowcharts which are:
(1) Logic Flowcharts
(2) Detail Flowcharts
Both of these flowcharts have the same format and are written by using almost the same symbols.

(1) Logic Flowcharts

The overall task to be executed is roughly considered in flowchart preparation as follows.
Data entry → Computation → Output as shown in the figure on the right. A logic flowchart functions as a preliminary step before preparing a detail flowchart, and is also used for a simple program preparation (coding).

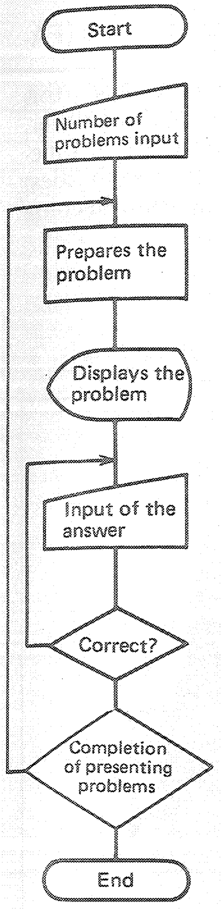

Start
Input of A and B
Computation of A ∗ B
Computation Result
End

(2) Detail flowcharts

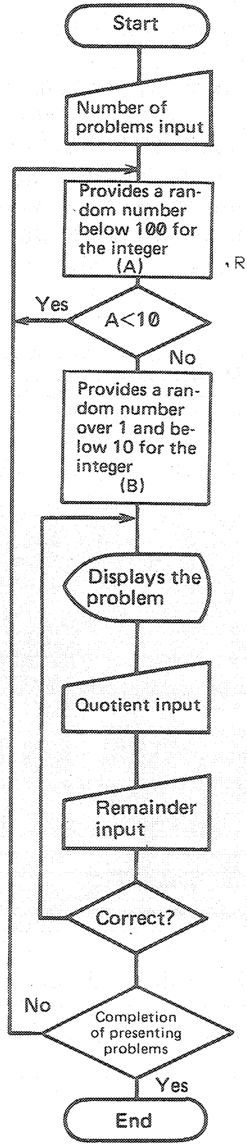
A detailed execution sequence of a task is written in the form of instructions according to a logic flowchart. However, a detailed flowchart is not required for a BASIC program.
Either way, it is important to take the time to write a flowchart before preparing a program with the sequence for the setup procedure, input mothod, processing method, and output format.

As an example, a mathematical drill program in which a 2 digit integer is divided by a 1 digit integer is prepared based on a flowchart.

[Logic Flowchart]


Start
Number of problems input
Prepares the problem
Displays the problem
Input of the answer
Correct?
Completion of presenting problems
End

[Detail Flowchart]


Start
Number of problems input
Provides a random number below 100 for the integer (A)
Yes
A<10
No
Provides a random number over 1 and below 10 for the integer (B)
Displays the problem
Quotient input
Remainder input
Correct?
No
Completion of presenting problems
Yes
End

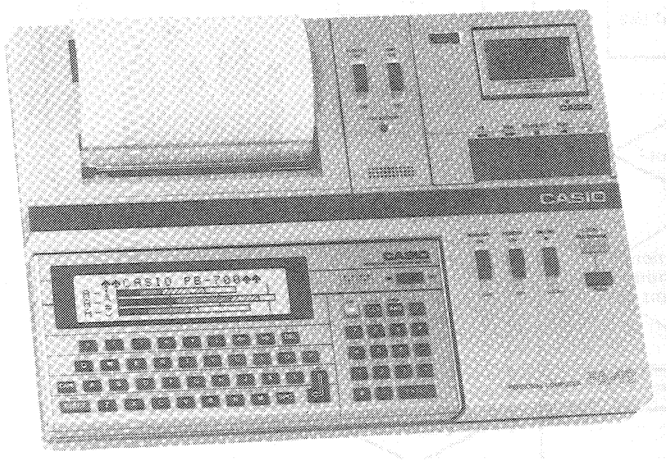
[Program]

```
 10 CLS
 20 INPUT "NUMBER OF PROBLEMS";N
 30 FOR I=1 TO N
 40 A=INT(RND*100)
 50 IF A<10 THEN 40
 60 B=INT(RND*9+1)
 70 CLS :PRINT "(";I;" )"
 80 PRINT TAB(5);A;" /";B;"= ";
 90 INPUT "",X
100 PRINT TAB(7);:INPUT " REMAINDER= "
,R
110 IF X<>INT(A/B) THEN 80
120 IF R<>A MOD B THEN 80
130 NEXT I
140 END
```

3-19 PB-700 GRAPHIC FUNCTIONS

The PB-700 has a large liquid crystal display (LCD) which displays 20 characters x 4 lines, and also has 160 x 32 dots which allow graphic displays.
PB-700 graphics can draw a precision graph and patterns by simple commands.
Also, a 4 color 114 mm wide plotter-printer with cassette interface (FA-10) can be connected to the PB-700.
Since up to 80 characters can be printed on 114 mm wide paper, this printer can be used in almost the same way as a full-scale plotter printer.
Since the compact PB-700 is provided with sophisticated graphic functions, please master these so that you can fully utilize its capabilities.
Although you may feel it is troublesome at first, you will soon become accustomed to the graphic functions as you use them.

Plotter-printer with cassette interface (FA-10)



3-20 GRAPHIC COMMANDS AND SCREEN COORDINATES

Since drawing patterns on the screen means the continuous connection of dots, this can be performed by drawing dots at specific locations.
The PB-700 is provided with the following graphic commands which draw or erase dots for the graphic display.

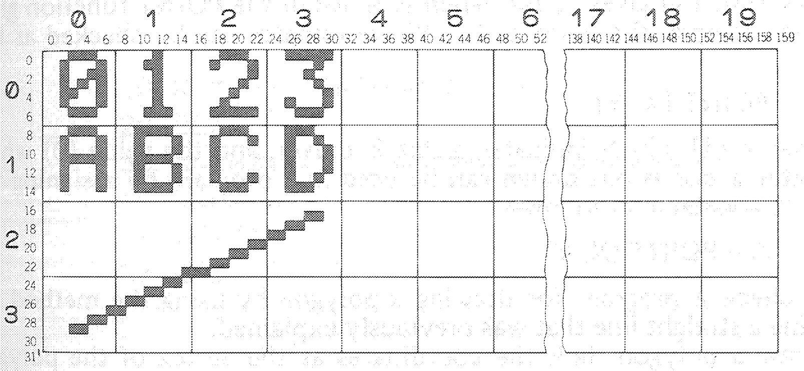
DRAW . . . . A command that draws a dot and a straight line.
DRAWC . . . A command that erases a dot and a straight line.

Also, the following convenient function is provided for graphics.

POINT . . . . Shows whether a dot is drawn or not at a specified location.

It is necessary to understand the dot locations (coordinates) on the screen before explaining the use of above commands and function.
Small numerals (horizontal: 0—159, vertical: 0—31) indicate the dot locations (graphic coordinates) as shown in the following figure. Large numerals (horizontal: 0—19, vertical: 0—3) indicate the character coordinates.
A character is considered to be graphics drawn by 8 x 8 dots, and its display location has to be determined.
Conversely, dots and lines can be drawn at any location with graphic coordinates.
The straight line from (3, 29) to (29, 17) in the following figure was drawn by a graphic command which allows graphics to be freely drawn anywhere.

Screen coordinates



Graphic coordinates consist of 5,120 dots with 160 dots in the X direction and 32 dots in the Y direction, in which the top left corner of the screen is (0, 0), and the bottom right corner is (159, 31) as shown in the figure on the last page.
Dot locations on the screen can be specified by these coordinates.
For example, to draw a dot at the (X, Y) location, use

DRAW (X, Y)

and to erase a dot at (X, Y), use

DRAWC (X, Y)

A straight line can be drawn with the same command by specifying the coordinates of both ends (X_1, Y_1) and (X_2, Y_2) of the line as follows.

DRAW (X_1, Y_1) — (X_2, Y_2)

A straight line from (X_1, Y_1) to (X_2, Y_2) can be drawn by this.
A line that connects three dots (X_1, Y_1), (X_2, Y_2) and (X_3, Y_3) can be drawn by specifying the following.

DRAW (X_1, Y_1) — (X_2, Y_2) — (X_3, Y_3)

Also, many continuous lines can be drawn by linking coordinates with "—".
A straight line can be erased by specifying the following.

DRAWC (X_1, Y_1) — (X_2, Y_2)

When a dot at a specified location is lit, or in other words, is drawn, the POINT function gives 1, and when it is not lit the POINT function gives 0. For example, the point of (X, Y) coordinates can be checked as follows.

POINT (X, Y)

The value (1) which indicates a dot is drawn, and the value (0) which indicates a dot is not drawn can be used in a program by assigning the value to a variable as follows.

A = POINT (X, Y)

Let's check a program for drawing a polygon by using the method for drawing a straight line that was previously explained.
To draw a polygon, link the coordinates at the vertex of the polygon with the DRAW command.

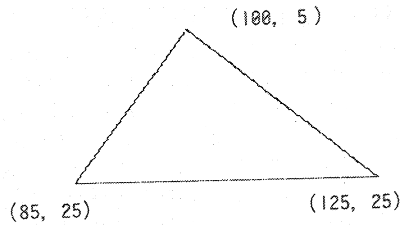
■ A Program That Draws A Triangle

A program that draws a triangle with the vertexes (100, 5), (85, 25), (125, 25) is as follows.

```
10 REM --- TRIANGLE ---
20 CLS
30 DRAW(100,5)-(85,25)-(125,25)-(100,
   5)
40 END
```

Coordinate specification by the DRAW command can also be performed with a numerical expression.
For example, the program above can be rewritten by using a numerical expression as follows.

Triangle drawn on the screen



```
10 REM --- TRIANGLE ---
20 CLS
30 X=100:Y=5 ...... Dot coordinates where drawing starts.
40 DRAW(X,Y)-(X-15,Y+20)-(X+25,Y+20)-
   (X,Y)
50 END
```

▪ A Program That Draws A Rectangle

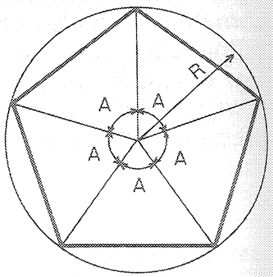
The following program draws a rectangle with the straight line (80, 5) – (150, 28) as its diagonal.

```
10 REM --- RECTANGLE ---
20 CLS
30 DRAW(80,5)-(150,5)-(150,28)-(80,28
   )-(80,5)
40 END
```

▪ A Program That Draws An Equilateral Pentagon

An equilateral polygon is one that is inscribed inside a circle with each vertex having equal spacing.
The figure on the right shows an equilateral pentagon that is inscribed inside a circle with an R radius. The lines that connect the 5 vertexes with the center of the circle all cross at the same angle, A (in this case A = 72°).
A program for a pentagon inscribed inside a circle with a center (100, 18), and a radius 15 can be drawn by using the above features as follows.

Equilateral Pentagon



```
10 REM --- PENTAGON ---
20 CLS
30 R=15            ......... Radius
40 X=100:Y=18      ......... Center of circle
50 A=360/5         ......... Angle
60 FOR I=0 TO 360 STEP A
70 DRAW(X-SIN(I)*R,Y-COS(I)*R)-(X-SIN
   (I+A)*R,Y-COS(I+A)*R)
80 NEXT I
90 END
```

▪ A program That Draws An Equilateral Polygon N

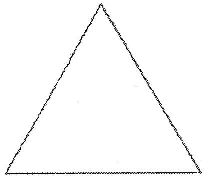
Equilateral polygon N can be freely drawn by changing angle A in the program for the pentagon.

Pentagon	A = 360/5
Polygon N	A = 360/N

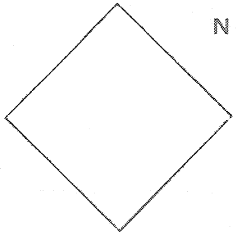
The following is a general purpose program that draws an equilateral polygon N by entering value of N.
However, the center of the circle is (100, 15).

```
 10 REM --- POLYGON ---
 20 CLS
 30 R=15              ......... Radius
 40 X=100:Y=15        ......... Center of circle
 50 INPUT "N=";N      ......... Equilateral polygon N
 60 A=360/N           ......... Angle of equilateral polygon N
 70 FOR I=0 TO 360 STEP A
 80 DRAW(X-SIN(I)*R,Y-COS(I)*R)-(X-SIN
    (I+A)*R,Y-COS(I+A)*R)
 90 NEXT I
100 END
```

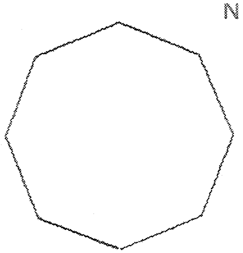
N = 3



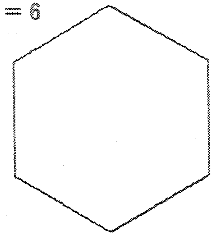
N = 4



N = 8



N = 6



3-21 DRAWING A CURVE

A curve can be drawn by specifying dot coordinates on the screen. However, the problem is how to specify the coordinates.
Many different curves can be drawn by using mathematical formulas. Let's draw a circle and SIN curve as follows.

■ A Program That Draws A Circle

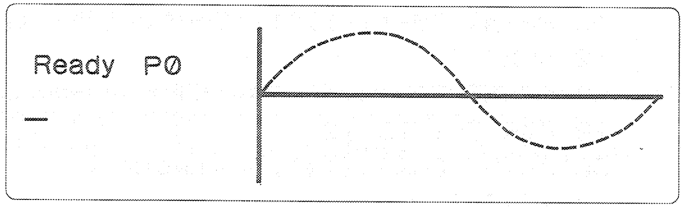
The pattern comes closer to that of a circle as the N of the previous program to draw an equilateral polygon is increased. Although a polygon cannot be a circle from a geometry viewpoint, a curve can be drawn by linking straight lines in the way previously mentioned to draw a curve on the screen.
The following program draws a circle with the coordinates (100, 15) as its center, and 15 as its radius.

```
10 REM --- CIRCLE ---
20 CLS
30 FOR I=0 TO 360 STEP 5
40 X=100+COS(I)*15 ········ Horizontal dot position
                            on the circular arc.
50 Y=15-SIN(I)*15  ········ Vertical dot position on
                            the circular arc.
60 DRAW(X,Y)
70 NEXT I
80 END
```

■ A Program That Draws A SIN Curve

The value of the SIN(I) function changes as 0 → 1 → 0 → −1 → 0 while the value of I changes from 0 to 360 degrees.
Therefore, a SIN curve can be drawn by drawing a dot, in which an appropriate enlargement is multiplied to the value of SIN(I), as a vertical position.
The following program also draws X and Y axes after drawing a SIN curve.

```
 10 REM --- SINE ---
 20 CLS
 30 A=65:B=15 ······ SIN curve origin (origin of the X, Y axes).
 40 M=12      ··········· Enlargement
 50 FOR I=0 TO 360 STEP 4
 60 X=A+I/4        ······················ X coordinate dot
 70 Y=B-SIN(I)*M   ··········· Y coordinate dot
 80 DRAW(X,Y)
 90 NEXT I
100 DRAW(A,2)-(A,28)     ········ Y axis
110 DRAW(A,B)-(A+92,B)   ········ X axis
120 END
```



```
Ready P0
_
```

3-22 DRAWING A LINE GRAPH

Although there are many different kinds of graphs, a line graph is drawn here as an example.
Line graphs are often used when time variations are checked, such as a temperature change for a certain period of time, stock price trends, etc. Therefore, it is recommended that the screen be fully utilized as much as possible so that the change can be clearly seen.

■ A Program That Draws The Monthly Average Temperature With A Line Graph

It is assumed that the monthly average temperature in a certain city is as shown in the following table. Data is entered in the program by a DATA statement, then the monthly average temperature is drawn by the line program while read is performed by a READ statement in the program.

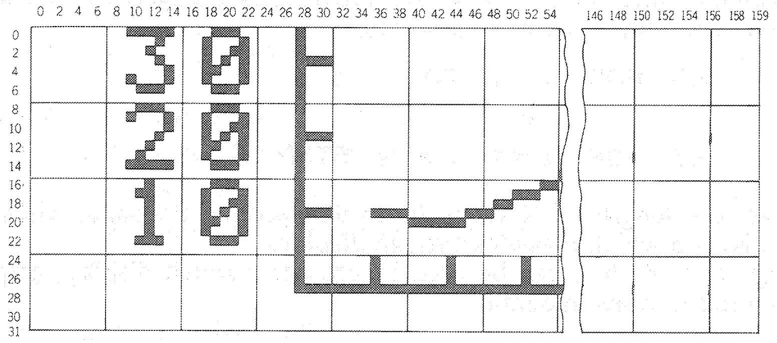
Month	Temp. (°C)	Month	Temp. (°C)	Month	Temp. (°C)
Jan.	11.5	May	19.8	Sep.	22.3
Feb.	9.8	June	23.4	Oct.	18.5
Mar.	13.7	July	26.6	Nov.	15.9
Apr.	18.3	Aug.	28.2	Dec.	14.7

```
 10 REM --- LINE GRAPH ---
 20 CLS
 30 FOR I=1 TO 3
 40 PRINT 30-(I-1)*10;CHR$(147)······CHR$(147)
                                             = ⊢
 50 NEXT I
 60 PRINT TAB(3);CHR$(154);······CHR$(154)= └
 70 FOR I=1 TO 12
 80 PRINT CHR$(144); ······CHR$(144)=┴
 90 NEXT I
100 FOR I=1 TO 12
110 X=36+(I-1)*8
120 READ A
130 Y=4+(30-A)*0.8
140 IF I=1 THEN 160
```

```
150 DRAW(P,Q)-(X,Y)
160 P=X:Q=Y
170 NEXT I
180 DATA11.5,9.8,13.7,18.3,19.8,23.4,2
    6.6,28.2,22.3,18.5,15.9,14.7
200 IF INKEY$="" THEN 200   Prevents the screen from
                            scrolling.
210 END
```

To visualize the locational relationship of the graph's values, vertical axis (Y), horizontal axis (X), and the lines, a part of the dot pattern on the screen is magnified. When you prepare a program for drawing a pattern or graph, it is recommended that a picture be drawn as mentioned above to determine the locational relationship.

Dot Pattern



In regard to a program technique, if there is not line 200, a command wait occurs soon after program execution has been terminated which causes the following display, and destroys the graph prepared with great effort.

```
Ready P0
```

Program execution is not terminated until a key is pressed by inserting the statemet of line 200, and the graph display remains as it is.

3-23 PREPARATION FOR DRAWING A BAR GRAPH

■ Drawing A Bar Graph By Using Characters

When you execute the following program and enter the numerals from 1 to 20 to N, N number of ▬ marks are displayed continuously.

```
10 CLS
20 INPUT "N=";N
30 FOR I=1 TO N
40 PRINT CHR$(131); ......CHR$ (131) is the ▬ mark.
50 NEXT I
60 END
```

A bar pattern is drawn with a length that is proportional to the numeral, N, which is the principle of a bar graph. Since the screen has only 20 digits, it is necessary to minimize the scale by one mark ▬ to display a magnitude that exceeds 20 digits.
For example, line 30 in the above program is changed as follows to display a maximum score of 100 points.

```
30 FOR I=1 TO N/6
          or
30 FOR I=1 TO N STEP 6
```

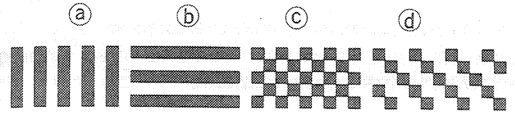
However, the length of N from 90 to 95 becomes the same when this method is used which provides a rough display.
Although this method can be used to provide a rough display, graphics shall be used to draw in details.

■ Drawing A Bar Graph With Graphics

The above program can be written with graphics as shown in the following program.
It can be displayed with 160 horizontal dots which is one-eighth the unit drawn by characters in this case.

```
10 CLS
20 INPUT "N=";N
30 FOR I=1 TO N*1.5
40 DRAW(I-1,8)-(I-1,14)
50 NEXT I
60 END
```

While the bars used for this program have a plain color, a program that is easy to see can be drawn by changing the pattern of the bars.
A bar graph can be drawn with the following pattern by adding or changing the following programs (a)—(d) in the above program.



```
ⓐ 35 IF I=N*1.5 THEN 40
  36 IF I MOD 2=0 THEN 50

ⓑ 30 FOR I=8 TO 14 STEP 2
  40 DRAW(0,I)-(N*1.5-1,I)

ⓒ 35 FOR J=8 TO 14 STEP 2
  40 DRAW(I-1,J+(I+1) MOD 2)
  45 NEXT J

ⓓ 25 FOR J=8 TO 14
  30 FOR I=1 TO N*1.5 STEP 3
  35 X=(J-8) MOD 3+I-1
  36 IF X>N*1.5-1 THEN X=N*1.5-1
  40 DRAW(X,J)
  50 NEXT I:NEXT J
```

3-24 TWO EXAMPLES OF BAR GRAPH PROGRAMS

Bar graphs are often used to visualize the relative relationship of data such as the results for students or salesmen, production amount, sales for different articles, etc.
However, since it is necessary to display the magnitude for each data item correctly with the length of a bar, as well as a relative comparison, it is important to draw with a proper scale that is within the limited range of the screen.

There are many ways to draw bar graphs depending on the application. Basically, there is a bar graph in which the quantity of a single item is indicated by one bar (such as when the amount for different products is indicated), and another in which the overall quantity and details are indicated by one bar at the same time (such as when the ratio of the total amount and the products is indicated).
Let's prepare programs which draw these two kinds of bar graphs by using the following example.

Data example: **Production amounts for two different cars A and B of an automobile manufacturer are shown in the following table.**

	1980	1981	1982
Automobile A	48,200	57,200	67,200
Automobile B	39,200	31,100	27,500

▨ : Pattern of the bar graph for Automobile A.

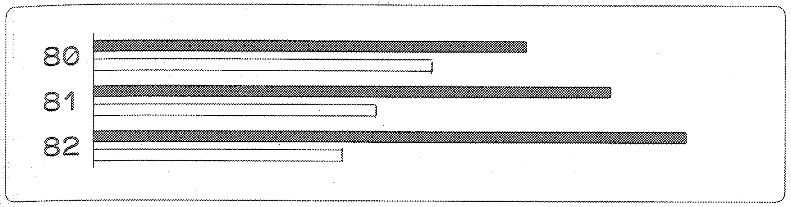
☐ : Pattern of the bar graph for Automobile B.

■ A Bar Graph Program In Which The Quantity Of A Single Item Is Indicated By One Bar.

```
10 REM --- BARGRAPH ---
20 CLS
30 FOR I=0 TO 2
40 LOCATE 0,I
50 PRINT 80+I;CHR$(136)  ······CHR$(136)=▌
60 FOR J=1 TO 2
70 READ A
80 FOR K=0 TO 2
```

```
 90 Y=I*8+(J-1)*4+K
100 IF J=2 THEN IF K=1 THEN DRAW(24+A/
    500,Y):GOTO 120
110 DRAW(25,Y)-(24+A/500,Y)
120 NEXT K:NEXT J:NEXT I
130 IF INKEY$="" THEN 130
140 END
150 DATA48200,39200,57200,31100,67200,
    27500
```

Display Example

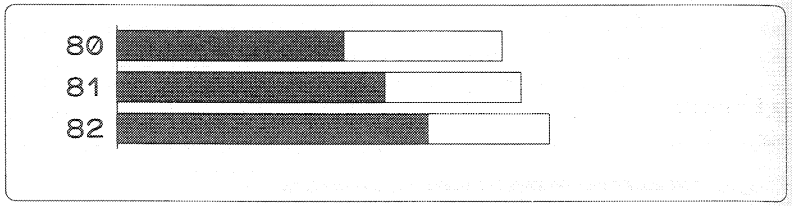


■ A Bar Graph Program In Which The Quantity And Details Are Indicated At The Same Time By One Bar.

```
 10 REM --- BARGRAPH ---
 20 CLS
 30 FOR I=0 TO 2
 40 LOCATE 0,I
 50 PRINT 80+I;CHR$(136)
 60 READ A,B
 70 FOR J=0 TO 6
 80 Y=I*8+J
 90 DRAW(25,Y)-(24+A/1000,Y)
100 NEXT J
110 X=24+A/1000:Y=I*8
120 DRAW(X+1,Y)-(X+B/1000,Y)-(X+B/1000
    ,Y+6)-(X+1,Y+6)
```

```
130 NEXT I
140 IF INKEY$="" THEN 140
150 END
160 DATA48200,39200,57200,31100,67200,
    27500
```

Display Example



3-25 ANIMATION DRAWING

■ Making Pictures Move

When you execute the following program, the * mark moves to the left and right.

```
 10 CLS
 20 FOR I=5 TO 15 ┐
 30 LOCATE I,1    │ Moves from left to right.
 40 PRINT " *"    │
 50 NEXT I        ┘
 60 FOR I=15 TO 5 STEP -1 ┐
 70 LOCATE I,1            │ Moves from right
 80 PRINT "* "            │ to left.
 90 NEXT I                ┘
100 GOTO 20
```

While the control variable I is changed from 5 to 15 on line 20, the cursor location is changed from (5, 1) to (15, 1) by the statement on line 30 as shown in the program. The * mark is displayed by drawing "␣*" at that position while the * mark previously drawn is erased by a space.

It seems as if it is moving from left to right by repeating this procedure. Also, lines 60—90 are the statements that move it from right to left.

The principle mentioned above is that of graphic animation which visually appears to be moving by combining drawings and erasing them. To move vertically is theoretically accomplished the same way. However, drawing and erasing cannot be simultaneously performed and the program becomes as follows.

```
10 CLS
20 FOR I=0 TO 3               ┐
30 LOCATE 10,I:PRINT "*";     │
40 IF I=0 THEN 60             │ Moves from up to down.
50 LOCATE 10,I-1:PRINT " "    │
60 NEXT I                     ┘
```

```
 70 FOR I=3 TO 0 STEP -1
 80 LOCATE 10,I:PRINT "*";
 90 IF I=3 THEN 110
100 LOCATE 10,I+1:PRINT " "
110 NEXT I
120 GOTO 20
```

Moves from down to up.

■ **Changing The Speed**

If the * mark moves too fast in the above program, the speed is controlled by using a FOR-NEXT statement.
Execute the program after adding the following line to the program on the last page.

```
45 FOR J=1 TO 50:NEXT J
```

When this statement is used, the speed of the movement from left to right becomes slightly slower. The speed is increased by reducing the final value of the FOR-NEXT statement, and is reduced by increasing the value.
The speed can be controlled by appropriately adding this statement to the necessary part.

■ **Moving A Dot Like A Curved Line**

The movement of a dot like a curved line becomes smoother by using graphic coordinates.
In the following program, a dot repeatedly moves on the locus of a dotted line inside a frame as shown by the Execution Example.

```
10 CLS
20 DRAW(13,0)-(13,31)-(137,31)-(137,0)
```

······Draws a frame.

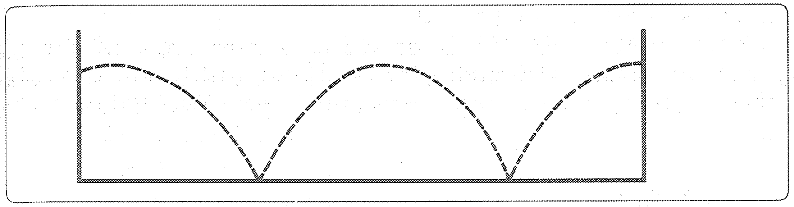
```
30 DRAW(14,0)-(14,30)-(136,30)-(136,0)
```

······Draws a frame.

```
 40 N=1:P=14
 50 FOR I=0 TO 360 STEP 3
 60 X=P+N:Y=29-25*ABS(COS(I))
 70 DRAW(X,Y)
 80 IF P=14 THEN 100
 90 DRAWC(P,Q)
100 P=X:Q=Y
110 NEXT I
120 N=-N
130 GOTO 50
```

······Computation of dot coordinates of the curved line.

Execution Example



■ **POINT Function Utilization**

The POINT function checks to see whether a dot is drawn or not at a specified location of the graphic coordinate.
If a dot is positioned at a horizontal coordinate (X) and a vertical coordinate (Y) which is (X, Y),

POINT (X, Y) = 1

If a dot is not at that position,

POINT (X, Y) = 0

The following program provides an example of POINT function utilization. At first, "CASIO PB-700" is displayed on the first line, then the POINT function checks whether each dot is drawn or not, and copies it to the third line depending on a value of 0 or 1 which is given.

```
 10 CLS
 20 PRINT "*** CASIO PB-700***"
 30 FOR X=0 TO 159
 40 FOR Y=0 TO 7
 50 IF POINT(X,Y)=0 THEN 70
 60 DRAW(X,Y+16)
 70 NEXT Y
 80 NEXT X
 90 IF INKEY$="" THEN 90
100 END
```

Lines 30–80 check whether dots are drawn or not on the first line of character coordinate, and if a dot is not drawn, line 50 causes the program to jump to line 70. If a dot is drawn, a dot is written at the same location on the third line by line 60.
If the plotter-printer (FA-10) is provided, a hard copy of the screen display can be made. Although plotter-printer utilization is explained in another section, a hard copy program is provided below for your reference.

```
 10 CLS
 20 PRINT "CASIO PB-700"
 25 LPRINT CHR$(28)+CHR$(37)
 30 FOR X=0 TO 159
 40 FOR Y=0 TO 7
 50 IF POINT(X,Y)=0 THEN 70
 60 LPRINT "D";X;",";-1*Y;",";X+0.4;",
    ";-1*(Y+0.4)
 70 NEXT Y
 80 NEXT X
 90 IF INKEY$="" THEN 90
100 END
```

3-26 GAME APPLICATIONS

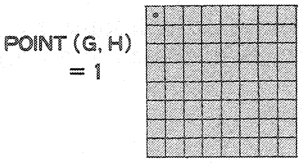
Animation is often used for games.
The following program is a block destroying game. Although sophisticated techniques are not specially required, a technique that shows a realistic change in the screen is required.
Although it is rather difficult for a beginner to do, try some contriving to slightly move the block position to the right, or move the racket to the left.
It is certain that you will make progress in preparing a game program with the technique mentioned above.
First input the following program.

```
 10 REM ---BLOCK DESTROY---
 20 CLS
 30 FOR I=0 TO 3
 40 LOCATE 2,I:PRINT CHR$(137);
 50 NEXT I
100 FOR I=3 TO 6
110 FOR J=0 TO 3
120 LOCATE I,J
130 PRINT CHR$(141);
140 NEXT J:NEXT I
200 LOCATE 17,1
210 PRINT CHR$(147)
220 R=1:S=0
300 FOR N=3 TO 1 STEP -1
310 LOCATE 0,0:PRINT N
320 X=10:Y=INT(RND*2+1):A=1:B=1
330 LOCATE X,Y:PRINT CHR$(236);
340 Q$=INKEY$
350 IF Q$="1" THEN GOSUB 500 ELSE IF Q
    $="0" THEN GOSUB 600
360 IF X=3 THEN A=-A
370 IF Y=0 THEN B=-B ELSE IF Y=3 THEN
    B=-B
```

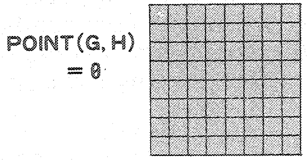
```
380 IF X=16 THEN GOSUB 800
390 LOCATE X,Y:PRINT " ";
400 IF X>16 THEN 440
410 G=(X+A)*8:H=(Y+B)*8
420 P=POINT(G,H)
430 IF P=1 THEN GOSUB 700
440 X=X+A:Y=Y+B
450 IF X>18 THEN BEEP :BEEP :BEEP  ELS
    E 330
460 NEXT N
470 CLS :LOCATE 2,1:PRINT "SCORE ";S
480 FOR I=1 TO 6:BEEP 1:NEXT I
490 IF INKEY$="" THEN 490 ELSE END
500 LOCATE 17,R:PRINT " ";
510 R=R-1
520 IF R<0 THEN R=0
530 LOCATE 17,R:PRINT CHR$(147);
540 RETURN
600 LOCATE 17,R:PRINT " ";
610 R=R+1:IF R>3 THEN R=3
620 LOCATE 17,R:PRINT CHR$(147);
630 RETURN
700 S=S+8-X:BEEP 1
710 LOCATE X+A,Y+B:PRINT " ";
720 X=X+A:Y=Y+B:A=-A
730 IF Y=0 THEN B=-B ELSE IF Y=3 THEN
    B=-B
740 RETURN
800 IF Y=R THEN A=-A:BEEP  ELSE IF Y+B
```

```
    =R THEN BEEP :A=-A ELSE 830
810 IF RND<0.7 THEN 830
820 LOCATE X,Y:PRINT " ";:X=X-1
830 RETURN
```

On line 420, the POINT function checks whether a block is drawn or not by checking whether a dot is drawn or not at the top left corner of a block as shown in the figure below.



A block drawn with CHR$ (141).



A block erased with CHR$ (32).

■ How To Play The Game

When you execute this program, a block and a racket are drawn, and a ball starts moving at a slant.
If the ball moves to the area on the right of the racket, one game is terminated. A game can be played three times.
The racket rises when "1" is pressed and descends when "0" is pressed.
After three games have been played, the score is displayed.

■ Variable Table Of Block Destroying Program

A,B	Moving direction of the ball
G, H	Coordinates which check whether a block is drawn or not
I, J	Variables for drawing blocks and fence
N	Number of remaining games to be played
P	Checks whether a block is drawn or not
Q$	Moving direction of the racket
R	The racket position
S	Score
X, Y	The ball position

3-27 DRAWING A PATTERN WITH THE PLOTTER-PRINTER

The plotter-printer with cassette interface (FA-10) can be connected to the PB-700.
Although the plotter-printer can be used as a character printer, it is convenient for drawing patterns because it can create patterns by lines which link one dot with another dot, while a dot printer prints characters or patterns by drawing dots.
The technique for drawing patterns is almost the same as that for the screen. However, since the plotter-printer draws patterns mechanically while the screen patterns are drawn electronically, it is necessary to increase its speed. Therefore, dedicated commands different from those for the screen are prepared for the plotter-printer as shown in the command table on the next page.
Since there are many commands, it seems troublesome to use them. However, the speed for drawing patterns is increased and programming problems are considerably reduced.

■ How To Use Commands

All commands for making graphics with the plotter-printer start with LPRINT. Many functions can be performed when commands shown in the command table are used together with LPRINT.
However, it is necessary to specify the graphic mode with the following statement before using these commands.

Graphic mode specification: LPRINT CHR$(28); CHR$(37)

The following statement is used to cancel the graphic mode (i.e., to specify the character mode in which characters can be printed).

Character mode specification: LPRINT CHR$(28); CHR$(46)

Plotter Command Table

	Command	Name	Description
For plotting use	O	ORIGIN	Origin specification of ORG coordinates
	D	DRAW	Links a dot with a dot specified by ORG coordinates.
	I	RELATIVE DRAW	Draws a line up to a specified dot.
	M	MOVE	Moves to a dot indicated by ORG coordinates with the pen up.
	R	RELATIVE MOVE	Moves to a specified dot with the pen up.
	A	QUAD	Draws a parallelogram on X and Y axes with the diagonal of 2 dots indicated by ORG coordinates.
	C	CIRCLE	Draws a circle and circular arc with a dot specified by ORG coordinates as the center.
	X	AXIS	Draws a coordinate axis from the origin of ORG coordinates toward +Y, +X, −Y, and −X directions.
	G	GRID	Draws horizontal and vertical stripes in a specified square.
	L	LINE TYPE	Draws a solid line, broken line, one-dot chained line, and two-dot chained line.
	B	LINE SCALE	Specifies the pitch of a broken line, one-dot chained line, and a two-dot chained line.
For character and symbol use	S	ALPHA SCALE	Specifies the size of characters and symbols.
	Q	ALPHA ROTATE	Specifies the rotary direction of characters and symbols.

	Command	Name	Description
For character and symbol use	Z	SPACE	Specifies character spacing for a following digit and line.
	Y	VERTICAL or HORIZONTAL	Specifies horizontal or vertical writing.
	P	PRINT	Prints a character string.
	N	MARK	Draws a mark with a pen position as its center.
For control use	J	NEW PEN	Pen color selection
	F	LINE FEED	Paper feed and paper return by one line unit.
	H	HOME	Modifies absolute coordinates, or shifts the pattern to a location that is easy to see.
	CHR$(64)	TEST	Checks pen use and pen ink. Execution Example: LPRINT CHR$(28) ; CHR$(37); CHR$(64) ⏎
For character control use	T	TAB	Tabulator
	?	FORMAT	Program list output use

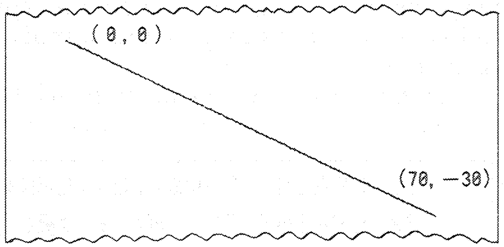
3-28 DRAWING A STRAIGHT LINE

■ Drawing A Line (D Command)

D is the command that draws a straight line.
Although the paper width of the plotter-printer is 114 mm, the pattern width that can actually be drawn is 95 mm.
When the power of the plotter-printer is turned on, the plotter pen is positioned at the left end. By assuming this point to be (0, 0) coordinate, the right end coordinate for the same line is (95, 0). By assuming that a line from (0, 0) to (95, 0) is the X axis, the part above this line is the plus coordinates of Y axis, while the part below this line is the minus coordinates of Y axis.
Let's draw a line from (0, 0) to (70, –30) by using the D command.

```
10 LPRINT CHR$(28);CHR$(37)… Graphic mode
                              specification
20 LPRINT "D";0;",";0;",";70;",";-30
              |        |        |        |
        X Coordinates  |  X Coordinates  |
                Y Coordinates      Y Coordinates
```

Execution Example



All characters and symbols that follow LPRINT, such as command "D" and commas "," shall be placed inside quotation marks (" "), and a semicolon ";" must be placed between each character or symbol as punctuation.
The instruction that draws a straight line from a dot (X_1, Y_1) to a dot (X_2, Y_2) is generally expressed as follows.

```
LPRINT "D";X1;",";Y1;",";X2;",";Y2
```

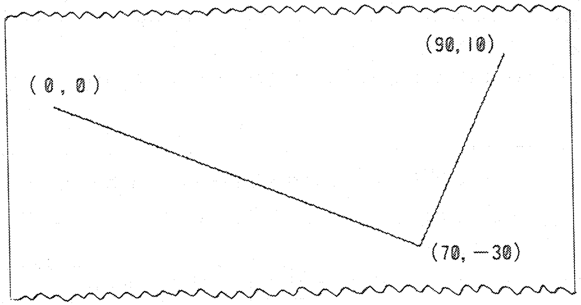
Note: When the parameter consists of numerical values, line 20 of the above program can also be written as follows.

```
20 LPRINT "D0,0,70,-30"
```

Let's draw two lines of (0, 0)–(70, –30) – (90, 10).

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "D";0;",";0;",";70;",";-30;
   ",";90;",";10
      (90. 10)    (0, 0)           (70, -30)
```

Execution Example



As many lines as desired can be drawn by specifying as many coordinates as required within the range of the number of characters (79 characters) as shown above.
Also, it can be written as follows by providing a line feed.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "D";0;",";0;",";70;",";-30
30 LPRINT "D";",";90;",";10
                     Draws a line from the present
                     pen location to (90, 10).
```

A line is generally drawn from the present pen position to (X2, Y2) by the following.

```
LPRINT "D";",";X2;",";Y2
```

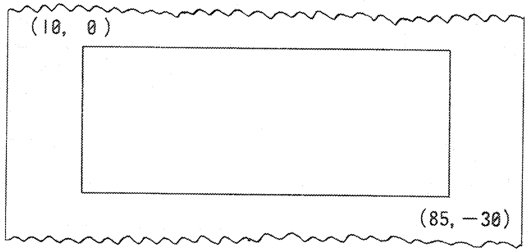
Since many quotation marks, commas, and semicolons are used, it seems to be troublesome. However, you won't feel it is troublesome as you become accustomed to using them. Also, they are not so hard to understand because they are repeated many times.

3-29 RECTANGLE, CIRCLE AND COLOR SPECIFICATION

■ Drawing A Rectangle (A Command)

Using the A command, the following rectangle was drawn by specifying both ends of diagonal, dot (10, 0) and dot (85, −30).

Execution Example



This program is prepared by using the A command as follows.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "A";10;",";0;",";85;",";-30
                 (10, 0)           (85, -30)
```

It can be drawn only by specifying both coordinates which are ends of the diagonal as shown above.

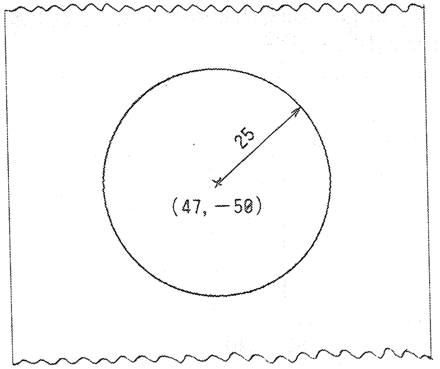
■ Drawing A Circle (C Command)

When the C command is used, a circle can be drawn only by specifying the center coordinates and radius.
A program that draws a circle with a radius of 25 and with a dot (47, −50) as the center is as follows.

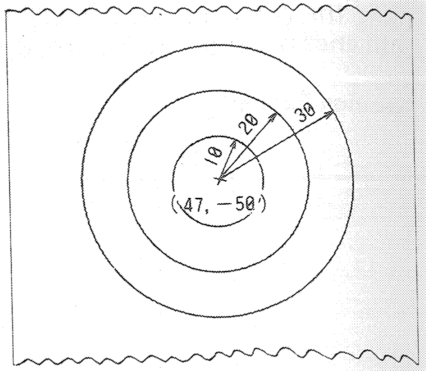
```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "C";47;",";-50;",";25
              Center coordinates   Radius
              of the circle
```

It is simpler than the program to draw a circle on the screen, isn't it?
An execution example of this program is shown by the following left figure.

Execution Example



Execution Example



Now, let's draw concentric circles with a radius of 10, 20 and 30, and center coordinates of (47, −50).
The program is as shown below while an execution example is shown by the above right figure.

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR R=1 TO 3
30 LPRINT "C";47;",";-50;",";R*10
40 NEXT R
```

Changes the radius.

■ Color Specification (J Command)

Pens with 4 different colors (black, blue, green, and red) are built into the plotter-printer, and they can be selected.
Execute the following program with line 25 added to the above program to draw concentric circles with color classifications.

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR R=1 TO 3
25 LPRINT "J";R
30 LPRINT "C";47;",";-50;",";R*10
40 NEXT R
```

The color is specified by the J command while each color can be specified with the following values.

LPRINT "J";0 ……… Black
LPRINT "J";1 ……… Blue
LPRINT "J";2 ……… Green
LPRINT "J";3 ……… Red

Also, when a numerical value is provided as shown above, they can be written as "J0", "J1", "J2", "J3". However, when a variable or numerical expression is used, the specification must be punctuated with semicolons (;).

3-30 TECHNIQUE FOR DRAWING A GRAPH

The plotter-printer shows its best capability when drawing many kinds of graphs such as line graphs, bar graphs, circular graphs, etc.

■ Coordinate Determination (O, H, X Command)

Since a graph making requires coordinates, their location must be determined first.

```
LPRINT "O" ; X ; "," ; Y
```

The origin coordinates can be placed at location (X, Y) by the "O" command.

```
LPRINT "H" ; 20  or LPRINT "H" ; A  (A is a variable)
```

The origin location is shifted downward by 20 or by the value of A when this command is used.
After the origin has been determined by this command, the coordinates (X axis, Y axis) for the line or bar graph and the scales are drawn by using the X command.

```
LPRINT"X";0;",";5;",";10
LPRINT"X";1;",";5;",";16
```

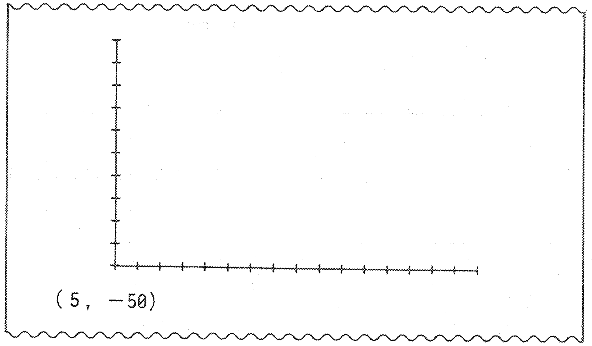
The numeral just after X indicates the axis direction.

0 : Draws Y axis from the origin upward.
1 : Draws X axis from the origin to the right.
2 : Draws Y axis from the origin downward.
3 : Draws X axis from the origin to the left.

The next numerical value (5) specifies the spacing of the scales (Unit: mm), and the next numerical values (10 and 16) indicate the number of scales.
This command is used as an example in the following program.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "O5,-50"   ······ Coordinate origin specification.
30 LPRINT "X0,5,10"  ······ Draws Y axis and scales.
40 LPRINT "X1,5,16"  ······ Draws X axis and scales.
```

When you run this program, the following coordinate axes are drawn with (5, −50) as the origin.



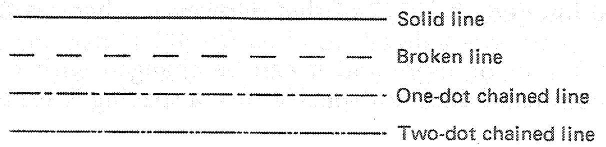
■ Changing The Kind Of Line (L, B Commands)

When many different kinds of data are handled by a graph, it is necessary to use some method for data classification.
One method is using different colors as previously mentioned.
Although this is very effective, they cannot be distinguished when a copy is made.
In this case, the kind of line, and bar pattern (when a bar graph is used) can be changed.

(1) Specifying The Kind Of Line (L Command)

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR I=0 TO 3
30 LPRINT "L";I
40 LPRINT "D20,";-I*5;",70,";-I*5
50 NEXT I
```

When you execute this program, four different kinds of lines are drawn as follows.



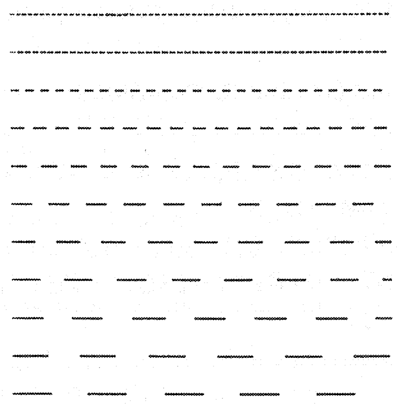
Values 0—3 after L indicate the following lines.

"L" ; 0 ········ Solid line
"L" ; 1 ········ Broken line
"L" ; 2 ········ One-dot chained line
"L" ; 3 ········ Two-dot chained line

(2) Changing The Spacing Of Broken Lines And Chained Lines (B Command)

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR I=0 TO 10
30 LPRINT "L1":LPRINT "B";I
40 LPRINT "D20";",";-I*5;",";70;",";-
   I*5
50 NEXT I
```

When you run this program, 11 lines are drawn which have different spacings.



The value of I specified by the B command indicates the spacing (Unit: mm) and the line length is 1/2 of that. However, when I = 0, the spacing is 0.5 mm.
A spacing is 1 mm or more and it can be changed with 0.2 mm units. When the B command does not specify this, a spacing is set to 6.4 mm.

■ Drawing Stripes (G Command)

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR I=0 TO 2
30 LPRINT "M20";",";0
40 LPRINT "G";I;",";50;",";-20
50 LPRINT "A20";",";0;",";70;",";-20
60 LPRINT "H10"
70 NEXT I
```

The G command is used on line 40 of this program.
The variable I is changed from 0 to 2 which specifies three different patterns as shown below.

0 : Plain color
1 : Horizontal stripes
2 : Vertical stripes

The succeeding numerical values 50 and —20 specify the range (50 mm to the right from the present pen position, 20 mm downward) in which the pattern is drawn.
The stripe spacing becomes 2 mm when ; " , "; 2 is added after that as shown below.

```
LPRINT "G";1;",";50;",";-20;",";2
           ┬     ┬       ┬       ┬
        Kind of 50 mm   20 mm   2 mm
        pattern to right downward spacing
```

Bar graphs are easier to see when different patterns are specified as shown above.

■ Moving The Pen (M Command)

A new command "M" appears on line 30 of the above program, it is also explained as follows.

```
LPRINT "M" ; 20 ; "," ; 0 or "M20,0"
```

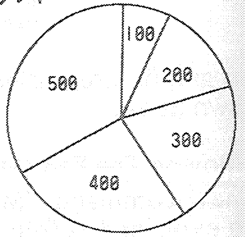
When the M command is written as shown above, it provides an instruction to move the pen to (20, 0) without drawing anything.

■ Drawing A Circular Graph (C Command Application)

The basic methods for drawing line graphs and bar graphs have been explained. The method for drawing circular graphs is explained below. The C command is used to draw a circle as previously explained. However, the lines to provide sectors must be drawn for a circular graph to match the data size.
A program which separates a circle into 100, 200, 300, 400 and 500 portions is shown below as an example.

```
 10 LPRINT CHR$(28);CHR$(37)
 20 LPRINT "C48,-50,30"
 30 S=0:T=0
 40 FOR I=1 TO 5:READ A:T=T+A:NEXT I:R
    ESTORE :A=0
 50 FOR I=1 TO 5:A1=A
 60 READ A:S=S+A
 70 X=48+INT(30*SIN(360*S/T)):Y=-50+IN
    T(30*COS(360*S/T))
 80 LPRINT "D48,-50,";X;",";Y
 90 A2=(A1+A)/2:GOSUB 200:NEXT I
 95 END
100 DATA100,200,300,400,500
200 X=50+INT(20*SIN(360*S/(T+A2))):Y=-
    48+INT(20*COS(360*S/(T+A2)))
210 LPRINT "M";X-10;",";Y
220 B$=STR$(A)
230 LPRINT "P";B$
240 RETURN
```

Execution Example



Total of the data is obtained on line 40, while the percentage for each data item and the X, Y coordinates on the circumference that divide the circle are obtained on lines 60—70.
On line 80, the lines are drawn from the obtained points (X, Y) to the center.

3-31 HOW TO WRITE GRAPHIC CHARACTERS

Although the basic purpose of graphics is to draw patterns, it is also important to draw characters in the patterns. Since many different commands are provided for this purpose, please utilize them fully.

■ Writing Characters (P, F Commands)

When characters are written in the graphic mode, the P command is used. The F command is used to perform line feed.
Next, execute the following program.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "P";"Alphabet"
30 LPRINT "F";2
40 FOR I=1 TO 26         )
50 LPRINT "P";CHR$(I+64) }... Writes 26 alphabetical
60 NEXT I                )    characters.
```

When the characters "Alphabet" are written as they are shown on line 20, they must be placed in quotation marks the same as for a PRINT statement. Of course, when the P command is used as shown on line 50, quotation marks are not required. "2" placed after the F command on line 30 indicates two line feeds. Execute this program.

Execution Example

```
Alphabet


        ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Although two line feeds are performed, 26 alphabetical characters are written from the location next to "Alphabet" because the F command only provides a line feed and does not include a return instruction.
Let's execute the same program again by adding the following line.

```
35 LPRINT "H"
```

As a result, line feeds and returns are performed as follows.

Execution Example

```
Alphabet


ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

The H command is used to move the origin of the Y axis downward as previously mentioned. However, if the movement distance is omitted as shown above, it returns to the origin.
However, since two line feeds are performed on line 30 by the F command, only the X axis returns to the origin. It is convenient to understand the combined utilization of the F and H commands.
The F command can also specify the following the same as before except when a variable or numerical expression is used.

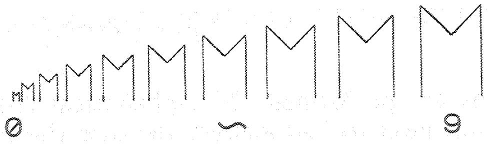
LPRINT "F2" .... Two line feeds

■ Specifying The Size Of A Character (S Command)

Ten different sizes from 0 to 9 can be specified for a character.
The character "M" can be written in sizes from 0 to 9 as follows.

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR I=0 TO 9
30 LPRINT "S";I
40 LPRINT "P";"M"
50 NEXT I
```

Execution Example



The relationship between sizes from 0 to 9 of the S command and the size of a character is shown in the table on the next page.

S Command Sizes And Character Sizes

S Size	0	1	2	3	4	5	6	7	8	9
Height (mm)	1.2	2.4	3.6	4.8	6.0	7.2	8.4	9.6	10.8	12.0
Width (mm)	0.8	1.6	2.4	3.2	4.0	4.8	5.6	6.4	7.2	8.0

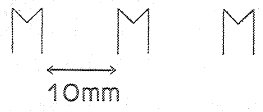
If a character is written by the P command without specifying the size by the S command, a size of "1" is specified.
The spacing between characters is 1.5 times the character width.

■ Specifying The Spacing Between Characters (Z Command)

The Z command is used to specify the spacing between characters.

```
10 LPRINT CHR$(28);CHR$(37)
20 FOR I=1 TO 3
30 LPRINT "S";4
40 LPRINT "Z";10
50 LPRINT "P";"M"
60 NEXT I
```

If 10 is specified as a value for Z command, as shown in this program, the characters are written with a spacing of 10 mm.



The length unit is specified in mm as shown above. However, when characters are written vertically, the spacing becomes the specified length +1 mm which will be explained later.

■ Vertical Writing Specification (Y Command)

The Y command specifies horizontal and vertical writing.

Horizontal writing ...... "Y0" or "Y";0
Vertical writing .... "Y1" or "Y";1

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "S";2
```

```
30 FOR I=0 TO 1
40 LPRINT "Y";I
50 FOR J=1 TO 2
60 LPRINT "P";123
70 NEXT J:NEXT I
```

Execution Example

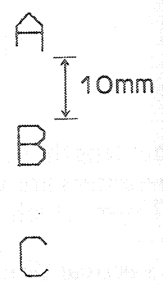
```
123 123
          1
          2
          3

          1
          2
          3
```

This program is executed as shown on the right.
The value of variable I is written by changing it to 0 and 1. Since 123 after the P command is treated as a numerical value, quotation marks are not used and one space for + sign is made.
The following program is used to write vertically with spacing between characters by using the Z command.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "S";2
30 LPRINT "Z";10
40 LPRINT "Y";1
50 LPRINT "P";"ABC"
```

Characters are written with a spacing of 10 mm by specifying "Z";10 as shown in the execution example on the right.

Execution Example



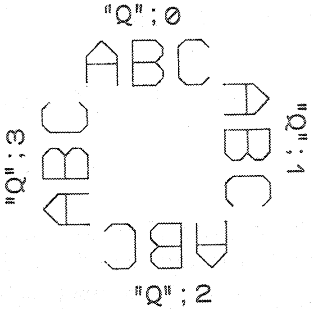
■ Specifying The Character Direction (Q Command)

The Q command specifies the direction of a character (up, down, left and right).
The direction is specified by the Q command and 0–3 which have the following relationship.

```
LPRINT "Q";0 or "Q0"·········Upward
LPRINT "Q";1 or "Q1"·········To the left.
LPRINT "Q";2 or "Q2"·········Downward
LPRINT "Q";3 or "Q3"·········To the right.
```

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "R";40;",";0
30 LPRINT "S";4
40 FOR I=0 TO 3
50 LPRINT "Q";I
60 LPRINT "P";"ABC"
70 NEXT I
```

The variable I on line 50 is changed from 0 to 3, and the characters ABC are written by line 60.
After writing the characters in one direction, they are written in the next direction from that pen position as shown on the right.



■ Pen Movement (R Command)

The R command that appeared on line 20 of the program in the previous section is explained as follows.

```
LPRINT "R";40;",";0
             :     :
          X axis  Y axis
        direction direction
```

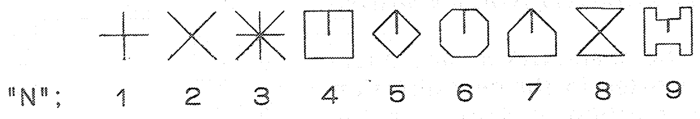
The R command is used to move the present pen position by the value of the X axis and Y axis direction.
This is slightly different from the M command which moves by specifying a coordinate location.
In the above example, the pen position is moved 40 mm from the present pen position to the right.

■ Writing Various Marks (N Command)

The N command can be used to write 9 different kinds of marks as follows by attaching a numerical value from 1 to 9 (the N command accepts 0, but no mark is given).
Since the size of a mark can be specified by the S command, the same as for a character, it can be used in many ways.

```
10 LPRINT CHR$(28);CHR$(37)
20 LPRINT "S";7
30 FOR I=0 TO 9
40 LPRINT "N";I
50 LPRINT "R";9;",";0
60 NEXT I
```

Execution Example



Please note that the pen stays at the same position after one mark has been written which is different from the way a character is written. Therefore, it is necessary to instruct the pen to move to a new pen position each time as shown on line 50.

3-32 DRAWING A SINE CURVE

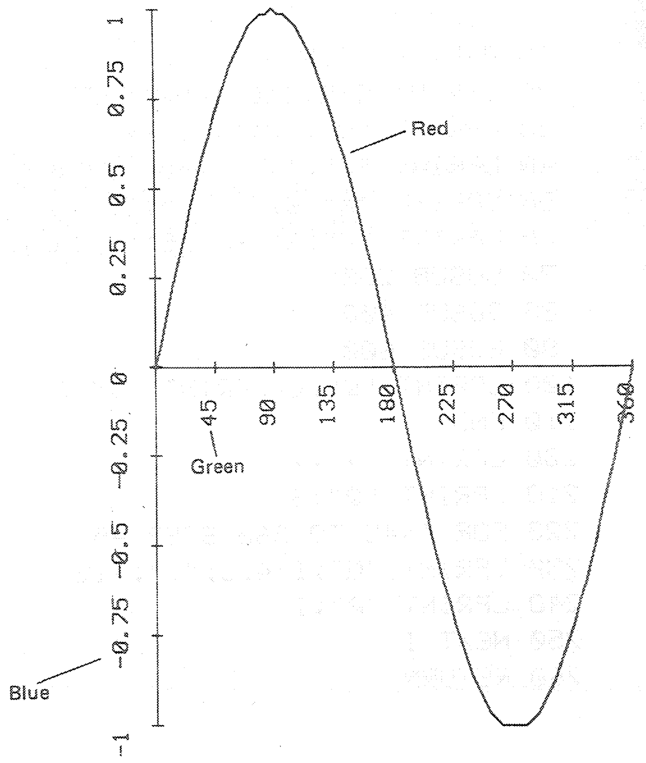
There are many other commands for the plotter-printer besides those which were explained. Please try many of them by referring to the Instruction Manual of the FA-10.
As a last exercise for the plotter-printer commands, a simple example of graphics is shown below.
This program is used to draw the X axis and the Y axis by the X command, to write the scale values by the P command, and to draw a sine curve.
The values of the scales can be placed in a vertical direction by the Q command, and they can be provided with many colors by the J command.
Also, all pen movement are performed by the M command.

Lines 40–60:	Draw X axis and Y axis (Black)
Lines 200–260:	Draw scales on X axis (Green)
Lines 400–450:	Draw scales on Y axis (Blue)
Lines 600–670:	Draw a sine curve (Red)

```
 10 REM --- SIN ---
 20 LPRINT CHR$(28);CHR$(37)
 30 LPRINT "O";10;",";-60
 40 LPRINT "X";1;",";10;",";8
 50 LPRINT "X";0;",";15;",";4
 60 LPRINT "X";2;",";15;",";4
 70 GOSUB 200
 80 GOSUB 400
 90 GOSUB 600
100 LPRINT "Q";0:LPRINT "J";0
110 END
200 LPRINT "J";2
210 LPRINT "Q";3
220 FOR I=45 TO 360 STEP 45
230 LPRINT "M";I/4.5;",";-12
240 LPRINT "P";I
250 NEXT I
260 RETURN
```

```
400 LPRINT "J";1
410 FOR I=-1 TO 1 STEP 0.25
420 LPRINT "M";-5;",";I*60-5
430 LPRINT "P";I
440 NEXT I
450 RETURN
600 LPRINT "M";0;",";0
610 LPRINT "J";3
620 FOR I=0 TO 360 STEP 4.5
630 X=I*(2/9)
640 Y=60*SIN(I)
650 LPRINT "D";",";INT(X);",";INT(Y)
660 NEXT I
670 RETURN
```

Execution Example



Preface to Chapter 4

The following terms are used in the "Format" section of each command described in Chapter 4.

- **"Numerical expression"**
  Equivalent to numerical expressions used in mathematics. Numerical operations are performed according to the precedence of the operators.
- **"Character expression"**
  Character strings can be concatenated by using the symbol "+". That is, addition of character strings makes a character expression.

  (Example) "ABC" + "DE" + "F" = "ABCDEF"

  Note that the number of characters of a character expression concatenated by + should be 79 or less.
- **"Variable"**
  Variables are used to store data. Since there are two types of data (numerical values and characters or symbols), we have numerical variables (such as A or B) and character variables (such as A$ or B$). Refer to Chapter 3 for details.
- **"Variable name"**
  A variable consists of an uppercase letter (A to Z) in its starting position and $, numbers, or uppercase letters that follow it.

  (Example) A, A$, AB, A1, XY$, X1$ . . . etc.

  A variable name means this format of a variable.
- **"Conditional expression"**
  A relational expression which compares the left side value and right side value by using a relational operator (such as =, = <, >, < >, etc.).
- **"Line number"**
  A number which is attached to the starting position of each program. Numbers 1 to 9999 can be used.
- **"Comment"**
  Comments are written to appropriate parts of a program in order to understand the content of the program.
  They do not affect the program execution at all.
- **"Message"**
  Messages are displayed on the screen in order to let you perform proper input, or to make the output easy to understand. They can be used by enclosing them with " ".
- **"File name"**
  When programs and data are transferred between the cassette tape and the computer, it is of great help if they are identified with names. A file name means such a name.

CHAPTER 4

COMMAND REFERENCE

COMMAND, STATEMENT AND BUILT-IN FUNCTION TABLE

Commands

- Cassette File Operation
  - BASIC Program Input/Output
    - Program Input — LOAD
    - Program Output — SAVE
    - Program Storing Check — VERIFY
- Program Execution
  - Program Execution — RUN
  - Continue Program Execution (Restart) — CONT
- Program Writing, Edit
  - Program Preparation
    - Specify Program Area — PROG
    - Program Erase — NEW
    - Program Area Status — SYSTEM
    - Initialize Memory — CLEAR
    - Delete Registered and Array Variable — ERASE
  - Program Edit
    - Program Delete — DELETE
    - Program Protect — PASS
    - Program Display — LIST
    - Line Correction Display — EDIT
- Output to Printer
  - Program Printing — LLIST
  - Output — LPRINT

Statements

- General
  - Assignment Statement
    - Variable Assignment — LET
  - Data Statements
    - Data Input — READ
    - Specify Data — DATA
    - Specify Data Statement Read Position — RESTORE
  - Array Statement
    - Define Array Variable — DIM
  - Branch, Control Statements
    - Unconditional Jump — GOTO
    - Subroutine — GOSUB ... RETURN
    - Conditional Jump — IF... THEN ... ELSE ...
    - Repeat Control — FOR ... TO ... STEP/ NEXT
    - Program Link Execution — CHAIN
    - Program Stop — STOP
    - Program Terminate — END
- Other Statements
  - Annotation — REM
- Graphic Statements
  - Screen Control Statements
    - Screen Erase — CLS
    - Cursor Location Control — LOCATE
  - Graphic Control Statements
    - Graphic Display (Dot Display) — DRAW
    - Graphic Delete (Dot Delete) — DRAWC

- Sound Statement
  - Bell — BEEP
- Input/Output Statements
  - Screen Output Statements
    - Output — PRINT
    - Format Control Output — PRINT USING
  - Input Statements from Keyboard
    - Input — INPUT
    - One Character Input — INKEY$
- File Input/Output Statements
  - File Input/Output
    - Output — PUT
    - Input — GET
- Trace Statements
  - Trace Mode Set — TRON
  - Trace Mode Release — TROFF

Built-In Functions

- General Functions
  - Screen Information
    - Dot On/Off — POINT
  - Format Control
    - Horizontal Position Control — TAB
- Numerical Functions
  - Mathematical Functions
    - Sine — SIN
    - Cosine — COS
    - Tangent — TAN
    - Arcsine — ASN
    - Arccosine — ACS
    - Arctangent — ATN
    - Specify Angle Unit — ANGLE
    - Exponential Function — EXP
    - Square Root — SQR

    - Natural Logarithm — LOG
    - Common Logarithm — LGT
    - Absolute Value — ABS
    - Give Integer — INT
    - Fractional Part — FRAC
    - Sign — SGN
    - Round — ROUND
    - Ratio of Circumference of Circle to Its Diameter — PI
    - Random Number Generation — RND
- Character String Functions
  - Character ↔ ASCII Code
    - ASCII Code — ASC
    - Character Conversion — CHR$
  - Character String ↔ Numerical Value
    - Numerical Value/Character String Conversion — STR$
    - Character String/Numerical Value Conversion — VAL
  - Character Extraction
    - Character String Left Part — LEFT$
    - Character String Extraction — MID$
    - Character String Right Part — RIGHT$
  - Character Position, Number of Characters
    - Character String Length — LEN

4-1 MANUAL COMMANDS

CONT

Function	To continue the execution of a program that was stopped by a STOP statement or STOP[ANS] key entry.
Format	CONT

Since program preparation is difficult, programs sometimes do not operate as expected even if they seem to have been completed. However, a program can be gradually completed by repeating debug. In this case, this CONT command is convenient.

When program execution has been stopped by the STOP command in a program or by operating the STOP[ANS] key, the CONT command is used to continue program execution. Program execution restarts from the line number next to the line that was stopped by the STOP command.

Usage

Insert the STOP command between sections for execution during program preparation to provide easy debugging.
The following program was prepared as an example.

```
10 READ R,H
20 S=PI*R^2
30 STOP
40 V=S*H
50 PRINT R,H,S,V
60 DATA10,20
70 END
```

When you execute this program with the RUN command, program execution is stopped at line 30 by the STOP command, and the display is as follows.

[CLS] [SHIFT] P0[0]

```
STOP P0-30
_
```

To check the execution content of line 10 and line 20, check the variables manually. To display the variable content, press R [ENTER], H [ENTER] and S [ENTER]. Then 10, 20 and 314.1592654 appear which proves that the execution has been performed correctly.
After checking has been completed, continue program execution by using the CONT command.

[SHIFT] CONT[W] [↵]

```
 6283.185307

Ready P0
```

Execution of line 40 and after is performed, and the value of the final result V is displayed and the execution terminates as shown above.
Debug can be easily performed by punctuating each program execution unit with a STOP command to check the variable contents.
When program correction is performed by specifying the EDIT mode while program execution is stopped by the STOP command, program execution is not continued by using the CONT command. In this case, continue program execution by operating RUN *line number.*

SEE STOP

DELETE

Function	Provides partial program deletion by line units.
Formats	DELETE *ln* DELETE *ln*– DELETE –*lm* DELETE *ln*–*lm* *ln* : Delete start line. *lm* : Delete end line. $(1 \leqq ln \leqq lm \leqq 9999)$

This command is used to delete a specific line in a program or lines in a specified range.
When DELETE is used, it has the following basic formats.

(1) DELETE line number	Deletes a specified line.
(2) DELETE line number –	Deletes lines from a specified line to the last.
(3) DELETE – line number . . .	Deletes all lines up to a specified line.
(4) DELETE line number *n* – line number *m*	Deletes the lines from line number *n* to line number *m*.

ln and *lm* have the following restriction.

$$1 \leqq ln \leqq lm \leqq 9999$$

Since this is a manual command, it cannot be used in a program.

Usage

A detailed explanation is provided by using the following program.

```
 10 REM DELETE SAMPLE
 20 PRINT "20:A"
 30 PRINT "30:B"
 40 PRINT "40:C"
 50 PRINT "50:D"
 60 PRINT "60:E"
 70 PRINT "70:F"
 80 PRINT "80:G"
 90 PRINT "90:H"
100 PRINT "END"
110 END
```

1 DELETE line number ⏎

For example, when line 50 in the above program is to be deleted, press the following keys.

DELETE 50 ⏎

Also, the same deletion can be performed by the following operation.

50 ⏎

2 DELETE line number – ⏎ (Use the ⊟ key to enter a "–".)

When the lines from line 70 to the last line are to be deleted in the above program, press the following keys.

DELETE 70 ⊟ ⏎

Also, line 70 and after are deleted by:

DELETE 65 ⊟ ⏎

If there is no line 65, line 70, which is closest to that number, and the following lines are deleted.

③ DELETE –line number ⏎

This command deletes the lines from the beginning to a specified line number. As an example, press the following keys.

DELETE ⊟ 40 ⏎

Check the listing with SHIFT LIST ⏎ , to determine that lines 10–40 were deleted. Also, exactly the same result can be realized by pressing the following keys.

DELETE ⊟ 42 ⏎

④ DELETE line number *n*–line number *m*

When the lines from line 30 to line 60 are deleted in the previous program, press the following keys.

DELETE 30 ⊟ 60 ⏎

Note: In this case, if the operation is performed as follows by reversing *n* and *m*, the entire program is destroyed. To prevent this, precautions should be taken.

DELETE 60 ⊟ 30 ⏎ The entire program is destroyed.

- The DELETE command cannot be used for programs with a password. This command cannot be executed for a program that was locked by the PASS command (see page 155), and a PR error is displayed.
- If a line includes a fraction, it is not executed, and an SN error occurs.

SEE NEW, NEW ALL

EDIT

Function	Allows a program to be modified.
Format	EDIT EDIT *ln* ($0 \leqq ln \leqq 9999$)

The EDIT command is used to edit a program for deletions, additions, corrections, etc.
The first line of a program is displayed by EDIT ⏎ , and the cursor appears at the end. Move this cursor to the position where a correction is to be made with ⇨ , ⇦ , SHIFT ⇧ or SHIFT ⇩ , and perform corrections with DEL or SHIFT INS .

- Using The Edit Keys

(1) ⇨ , ⇦	Moves the cursor to the right and left. If key depression is continuous, the cursor moves from the left end of the display to the end of a line by the repeat function.
(2) SHIFT ⇧ SHIFT ⇩	The cursor can be moved up and down by pressing the SHIFT key and the Cursor key at the same time.
(3) DEL	Deletes one character or symbol where the cursor is located, and fills the space.
(4) SHIFT INS	Makes one space between the cursor and the character or symbol just before the cursor.

Usage

Now let's edit a program by inputting the following program.

```
10 PRINT "AVERAGE"
20 INPUT A,B,C
30 H=(A+B+C)/3
40 PRINT H
50 GOTO 10
```

1 Checking A Program List From The Beginning

When a program list is displayed from the beginning and 1 line is corrected at a time, the following is used.

EDIT [↵]

Next, each time the [↵] key is pressed, the list will be continuously displayed with the smallest line number first.

2 Checking A Program List From The End Toward A Smaller Line Number (IN REVERSE)

When line 10 of a program is checked after a program listing from the beginning to line 50 has been completed for the previous program, perform the following operation.

Program list displayed up to line 50.

```
20  INPUT  A, B, C
30  H= (A+B+C) /3
40  PRINT  H
50  GOTO  10
```

⇩

Operate [SHIFT] [▲] four times until line 10 is displayed as shown on the right.

```
40  PRINT  H
30  H= (A+B+C) /3
20  INPUT  A, B, C
10  PRINT "AVERAGE"
```

3 Checking A Program From A Middle Line Number

When you check a line number listed in the middle of a program, perform the following operation.

EDIT line number [↵]

After this, the following lines are displayed every time the [↵] key is pressed.

4 Using [⇦], [⇨], [INS DEL]

Line 20 and line 30 in the program are corrected as follows.

```
20  INPUT  A, B, C, D
30  H=(A+B+C+D)/4
```

When you use EDIT [↵] to list line 20, the display is as follows.

[SHIFT] [EDIT E] 2 0 [↵]

```
Ready  P0
EDIT  20
   20  INPUT  A, B, C_
```

In regard to line 20, after entering " , " and then "D", the correction is completed by pressing the [↵] key with line 30 displayed at the same time as follows.

[,] [D] [↵]

```
Ready  P0
EDIT  20
   20  INPUT  A,B,C,D
   30  H= (A+B+C)/3_
```

In regard to inserting "+D" in line 30, provide space for two characters with:

[⇦] [⇦] [⇦] [SHIFT] [INS DEL] [INS DEL]

Pressed together

and press [+] [D] [⇨] [⇨] [4] [↵]

- If the EDIT command is used for a program with a password, a PR error will occur.
- Edit Mode Release
  The Edit Mode is released as follows.
  (1) When the BRK key is pressed.
  (2) When the CLS key is pressed.
  (3) When incorrect operation is performed.
  (4) When there is no program to be displayed.
  (5) When the power is turned off.

LIST/LLIST

Functions	LIST: Displays the content of a program. LLIST: Prints out the content of a program.
Formats	LIST(LLIST)V LIST(LLIST)ALL LIST(LLIST) LIST(LLIST)*ln* LIST(LLIST)*ln*– LIST(LLIST)–*ln* LIST(LLIST)*ln*–*lm* *ln*, *lm* : line numbers (1 ≦ *ln* ≦ *lm* ≦ 9999)

LIST and LLIST commands are used to confirm program lists stored in the PB-700. LIST displays lists on the display while LLIST prints out lists on roll paper.

Not only program lists, but programmed variables used as registered variables can be listed.

Usage

LIST and LLIST commands are utilized as follows.

(1) LIST (LLIST)V	Lists registered variables.
(2) LIST(LLIST)ALL	Lists all programs from P0 to P9.
(3) LIST (LLIST)	Lists the program in a specified program area.
(4) LIST(LLIST)line number ...	Lists only a specified line of a program in a specified program area.
(5) LIST(LLIST)line number–..	Lists from a specified line to the last of a program in a specified program area.
(6) LIST(LLIST)–line number..	Lists all lines up to a specified line of a program in a specified program area.

(7) LIST(LLIST)line number *n*—line number *m*
Lists the lines with line numbers from *n* to *m* of a program in a specified program area.

Next, enter the following program list to understand the utilization of the LIST and LLIST commands.

List

```
        10 PRINT "AVERAGE"
        20 INPUT "A=",AB,"B=",CD,"C=",A1
        30 D1=(AB+CD+A1)/3
        40 PRINT D1
        50 GOTO 10
```

* This program uses registered variables, although they are irregular, to show the utilization of LIST V.

[1] LIST(LLIST)V ⏎

This lists all presently used registered variables and array variables. To provide a print out, enter LLIST V ⏎ , then the registered variables used in the above program are printed as follows.

```
AB    CD    A1    D1
```

Also, for array variables specified with the DIM command, except for registered variables, the array variable names are displayed.

[2] LIST (LLIST) ALL ⏎

The PB-700 has program areas with a range from P0 to P9 into which independent programs can be entered.
When the entered programs in all program areas are to be displayed all at once, this command is used.
When you enter LLIST ALL ⏎ , all program lists are printed out as follows. This is performed in a sequence with the smallest line number first. Also, the program areas are always printed out.

```
  P0
    10 PRINT "AVERAGE"
    20 INPUT "A=",AB,"B=",CD,"C=",A1
    30 D1=(AB+CD+A1)/3
    40 PRINT D1
    50 GOTO 10
```

[3] LIST (LLIST) ⏎

When you enter this, all of a program in a presently specified program area is displayed (printed) in a sequence with the smallest line number first.

[4] LIST (LLIST) line number ⏎

When you specify a line number after the LIST (LLIST) command, the specified line of the program in a presently specified program area is displayed (printed). For example, when you enter PROG 0 ⏎ , LIST 30 ⏎ , the following is displayed.

```
30  D1 = (AB+CD+A1)/3
```

[5] LIST (LLIST) line number— ⏎

When you place "—" (minus key input) after a line number, the specified line and after of a program are displayed (printed) sequentially. For example, when you enter LIST 30 ⊟ ⏎, the following is displayed.

```
30  D1 = (AB+CD+A1)/3
40  PRINT D1
50  GOTO 10
```

6 LIST (LLIST)—line number ↵

When you place "–" before a line number, all the program up to the specified line number are displayed (printed) from the beginning. For example, when you enter LIST ⊟ 30 ↵, the following is displayed.

```
10  PRINT "AVERAGE"
20  INPUT "A=", AB, "B=", CD, "C=", A1
30  D1= (AB+CD+A1)/3
```

7 LIST (LLIST) line number *n*—line number *m* ↵

When this is entered, program with line numbers from *n* to *m* are displayed (printed). Because the program is displayed in a sequence with the smallest line number first, it is definite that $n \leq m$. For example, when you enter LIST 20 ⊟ 40 ↵, the following is displayed.

```
20  INPUT "A=", AB, "B=", CD, "C=", A1
30  D1= (AB+CD+A1)/3
40  PRINT D1
```

Technique

1 Halting LIST, LIST ALL Execution

Because the lines of a program are displayed continuously when a program is listed with the LIST command, it is difficult to confirm the program content. To overcome this, press the ↵ key to momentarily stop the display. To resume the program listing, press the ↵ key again.
LIST command execution cannot be stopped with SHIFT STOP ANS.
When a program listing check is not required, press the BRK key to cancel LIST or LIST ALL command execution.

2 Leaving LLIST, LLIST ALL

When a program list is being printed out with the LLIST or LLIST ALL commands, temporary suspension cannot be performed with the ↵ key. You can only cancel LLIST command execution by pressing the BRK key to stop the print out of the program list.

LOAD

Function	Loads a program stored on a cassette tape into the main frame memory.
Formats	LOAD LOAD ALL LOAD, A LOAD, M

The LOAD command is used to read a program that was stored on a cassette tape by a SAVE command back into the PB-700. The file name that was used during SAVE is now useful. Enter the file name together with the LOAD command, then the PB-700 automatically searches for the file name, and reads the program from the tape.
If a file name is not specified, the program found first is read in.

Explanation

The LOAD command has the following formats.

(1) LOAD
Reads the program found first, among those stored by SAVE or SAVE "file name", into the presently specified program area.
There is no problem even if the program area during "SAVE" and the presently specified program area are different.

(2) LOAD "file name"
Reads the program that was stored with the same file name into the presently specified program area. In this case, the program area during "SAVE" and the program area during "LOAD" can be different.

(3) LOAD ALL
Reads programs that were stored by SAVE ALL to the same program areas. Since there is no file name, the programs found first among those stored by SAVE ALL are read in.

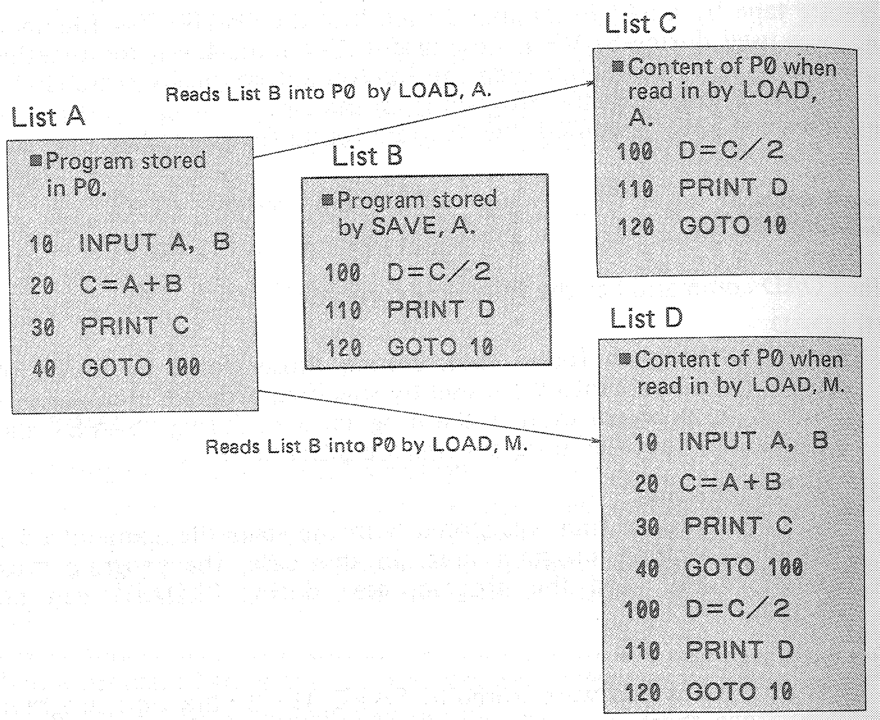
(4) LOAD ALL "file name"
Reads the programs with the same file name among those stored by SAVE ALL "file name" into the PB-700.

(5) LOAD, A/LOAD "file name", A

Reads a program that was stored with ASCII code (SAVE, A or SAVE "file name", A) into the PB-700. If there is no file name, the program found first is read in, and if there is a file name, the program with the same file name is read in.

(6) LOAD, M/LOAD "file name", M . . . . .

Reads a program that was stored by SAVE, A or SAVE "file name", A. LOAD, A or LOAD "file name", A is similar to this.



Reads List B into P0 by LOAD, A.
Reads List B into P0 by LOAD, M.

List A

■Program stored in P0.

```
10  INPUT A, B
20  C=A+B
30  PRINT C
40  GOTO 100
```

List B

■Program stored by SAVE, A.

```
100  D=C/2
110  PRINT D
120  GOTO 10
```

List C

■Content of P0 when read in by LOAD, A.

```
100  D=C/2
110  PRINT D
120  GOTO 10
```

List D

■Content of P0 when read in by LOAD, M.

```
10  INPUT A, B
20  C=A+B
30  PRINT C
40  GOTO 100
100  D=C/2
110  PRINT D
120  GOTO 10
```

●As soon as a program is loaded, NEW also generally functions to erase a previous program. However, if LOAD, M is used, the previous program will remain if the line numbers are different.

Usage

1 **Loads a program with the same format as that during SAVE.**

Some formats have a correspondence with each other such as LOAD is used for programs stored by SAVE, and LOAD ALL is used for programs stored by SAVE ALL. The combinations of LOAD and SAVE are as follows.

	LOAD	LOAD "file name"	LOAD ALL	LOAD ALL "file name"	LOAD,A	LOAD "file name"	LOAD, M.	LOAD "file name", M
SAVE	○	×	×	×	×	×	×	×
SAVE "file name"	○	○	×	×	×	×	×	×
SAVE ALL	×	×	○	×	×	×	×	×
SAVE ALL "file name"	×	×	○	○	×	×	×	×
SAVE, A	×	×	×	×	○	×	○	×
SAVE "file name", A	×	×	×	×	○	○	○	○

2 **Password**

If a program with a password is stored, the password is also stored. A reference for when programs with a password attached are loaded is as follows.

(1) When a password is not provided in the PB-700, LOAD can be performed. In this case, a stored password exists as a PB-700 password.

(2) When a password the same as that affixed to a stored program is provided in the PB-700, it can be stored as it is. In this case, a password exists.

(3) When a PB-700 password is different from that of a saved program, a PR error is displayed as soon as LOAD starts, and the cassette tape stops.

3 Error during LOAD

(1) RW error . . . . .
This error occurs when a parity error occurs during LOAD. In this case, clear the program, which has been loaded, by entering NEW ↵ and perform loading from the beginning.

(2) OM error . .
This error occurs when the memory capacity is shortened. In this case, clear unnecessary programs or expand RAM capacity.

SEE SAVE, VERIFY, CHAIN, PUT, GET

NEW/NEW ALL

Function	Program erase.
Formats	NEW NEW ALL

When new program input is performed, it is necessary to erase the previous program. The commands that erase the previous program are the NEW/NEW ALL commands.

1 NEW Command Functions

The NEW command erases a program in a specified program area with the PROG command used to specify the program area. When the NEW command is executed with a password (see page 155, PASS) attached, a PR error occurs.
Also, the clearing of variables cannot be performed (see page 168, CLEAR).

2 NEW ALL Command Functions

The NEW ALL command erases the programs in all of the program areas at one time. Since this command is effective when the PASS command has been executed, careful confirmation should be performed when it is used.
Also, it not only clears programs but also clears variables, and initializes the computer as follows.

(1) Clears all numerical variables to 0.
(2) Clears all character variables to " " (null-strings).
(3) All registered variables are released.
(4) Cancels DIM command (see page 170, DIM), and releases array declarations.

Also, the program area is set to P0, and ANGLE is set to 0, DEG (degrees). The NEW ALL command resets all RAM to the initial state as mentioned above.

Usage

NEW and NEW ALL commands cannot be used in a program. They can be used only by pressing keys directly as follows.

NEW ↵ or NEW ALL ↵

PASS

Function	Protects a program by assigning a program password.
Format	PASS "password"

Often a program that was prepared with great effort is erased by mistake, or is destroyed by writing another program on top of it. Also, it might be erased by another person for his own convenience.
Therefore, important programs are locked with this command to prevent program corrections and erasures from being performed.

Usage

1 PASS Command Utilization

The PASS command is utilized by entering:

PASS "Password with up to 8 characters" ↵

When this command is used, program corrections and erasures cannot be performed unless the password is cancelled. (This command cannot be used in a program.) Actually the following commands cannot be executed.

(1) DELETE
(2) EDIT
(3) LIST, LLIST
(4) NEW

Also, a program can not be newly written. The functions of the PASS command are effective in all of the program areas. Conversely, the PASS command cannot function in only a certain program area. If the commands listed above are executed when the PASS command has been executed, a PR error will be displayed.

2 PASS Command Release

Characters, numerals and symbols can be used for a password with up to 8 characters. To release a password, make an entry using exactly the same password as follows.

PASS "Password with up to 8 characters" ↵

Therefore, if a password that was entered has been forgotten, the PASS command function cannot ever be released. Since a password cannot be observed, it is recommended that something which cannot be forgotten, such as your name, should be used for a password. Also, before password entry is performed, it is important to confirm that it is correct. If passwords are forgotten, the only solution is to remove the batteries or to execute the NEW ALL command as an emergency countermeasure. However, since the programs in all the program areas will disappear, they should be stored on a cassette tape by using the SAVE command.

SEE SAVE, LOAD, NEW

PROG

Function	Specifies a program area.
Format	PROG numerical value (or numerical expression) $0 \leqq$ numerical value (numerical expression) < 10

The PB-700 is provided with 10 program areas (P0—P9) where independent programs can be written.
The PROG command is used to specify a program area.

Usage

When the PB-700 power is turned on, the display will be as follows.

Power ON

```
Ready P0
_
```

When the power is turned on, the program area P0 is automatically specified as shown above. Next, let's specify P9 as the next program area.

Power ON
SHIFT PROG 0 9 ↵

```
Ready P0
PROG 9
Ready P9
_
```

The specification has been terminated with program area P9 in an input wait status.
Though a numerical expression can be entered after PROG, the computation result (X) must be a value with the range of $0 \leqq X < 10$.
If it is outside this range, a BS error is displayed.

Examples

PROG (100/20) → Specifies P5 with PROG5.
PROG (100/10) → Displays a BS error.
PROG (100/15) → Specifies P6 (The fractional part is discarded.)

SEE GOTO, GOSUB

RUN

Function	Executes a program.
Formats	RUN RUN *ln* (0 ≦ *ln* ≦ 9999)

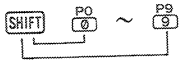
The RUN command is used to execute a program in the presently specified program area.

The RUN command has the following two formats.

(1) **RUN ⏎ .....**
Starts program execution from the first line of the specified program area. Execution is performed in a sequence with the smallest line number first.

(2) **RUN line number ⏎ .....**
Starts execution from the line number after RUN. If the specified line number does not exist, execution starts from the line closest to the specified line number which is larger than the specified line number.

Execution can be started from the beginning of a program in each program area as follows without using the RUN command.



Programs P0–P9 can be executed by pressing the SHIFT key and the 0 ~ 9 keys at the same time.

SAVE

Function	Stores a program on cassette tape.
Formats	SAVE SAVE ALL SAVE, A SAVE "file name" SAVE ALL "file name" SAVE "file name", A

It is fun to create your own program library by saving programs, completed with great effort, on cassette tape.
The SAVE command is used when a program is stored on cassette tape.
This command has the following formats.

(1) SAVE	Stores a program in a presently specified program area with a binary code format.
(2) SAVE ALL	Stores the programs of all program areas with a binary code format.
(3) SAVE, A	Stores a program in a presently specified program area with an ASCII code format.

- SAVE with an ASCII code format requires more time and uses more cassette tape compared to SAVE with a binary code format (internal code format). However, it is necessary to use SAVE with an ASCII code format to execute LOAD, M (see LOAD command).

Usage

It is recommended to provide a "file name" when a program is saved on cassette tape. Because it is very convenient when a plural number of programs are stored or when a program is read out from cassette tape to the PB-700.

1 File Name Placement

SAVE "file name"
SAVE ALL "file name"
SAVE "file name", A

- Although any character or symbol can be used as a file name, 8 characters or less must be used.
- In regard to a program with a password, password output is also performed at the same time.

2 Record Tape Counter Number

When you press the ↵ key with the cassette tape recorder in the record mode, recording starts and the tape automatically stops when it has terminated with the remote function activated. Make a record of the tape counter number for recording start and end. It is very convenient for saving the next program.

3 Correct Recording Confirmation

Confirm if cassette tape recording was correctly performed by using the VERIFY command (see page 163, VERIFY). If it is not correctly recorded, perform the recording again.

File Formats

The file format names of programs or data, which have been stored by SAVE or PUT, are displayed when they are loaded to the computer by LOAD or GET. The file format is determined by the way of SAVE or PUT.

Operation for output	Display during loading
SAVE	PF B
SAVE ALL	AF B
SAVE, A	PF A
SAVE "ABC"	ABC PF B
SAVE ALL "ABC"	ABC AF B
SAVE "ABC", A	ABC PF A
PUT A	DF A
PUT "ABC" A	ABC DF A

The file names are displayed as they are.
Meanings of the file formats are as follows.

Display during loading

```
PB-700     P F   B
File name (A)(B) (C)
```

(A) P : Program
A : All programs
D : Data

(B) File

(C) B : Binary (Internal code format)
A : ASCII (ASCII code format)

SYSTEM

Function	Displays the program area status, number of remaining bytes, and specified angle unit.
Format	SYSTEM

The SYSTEM command is used to display the internal status, the program area status, number of remaining bytes and specified angle unit, of the PB-700.
Enter SYSTEM ⏎ , and if there is no program and data is stored in the PB-700, the following is displayed.

```
P 0123456789
2864 Bytes:ANGLE 0
Ready P0
_
```

The SYSTEM command is used to provide display like this. What does this display mean?

(1) P 0123456789 on the first line indicates whether a program is written or not in the program areas P0 to P9.
For example, input 10 PRINT "CASIO PB-700" ⏎ . Then the display becomes as follows

SHIFT SYSTEM S ⏎

```
P ♥123456789
2846 Bytes:ANGLE 0
Ready P0
_
```

♥ indicates the program area where a program has already been written.

(2) On the second line, 2864 Bytes indicates the number of remaining bytes that can be used. (This number changes when the optional RAM expansion pack are installed.)

(3) Also, on the second line, ANGLE 0 indicates the angle unit. ANGLE 0 (DEGREE) is specified just after power is turned on.

(4) On the third line, Ready P0 indicates that program area P0 input can be performed in the program area P0. P0 is always specified just after the power is turned on.

VERIFY

Function	Performs a parity check of programs or data stored on a cassette tape.
Formats	VERIFY VERIFY "file name"

The VERIFY command checks if programs or data stored on a cassette tape by the SAVE or PUT command are correctly stored.

- This command has the following formats.

(1) VERIFY. . . . .
Checks the first program found on cassette tape.

(2) VERIFY "file name" . . . . .
Checks the program with the same file name on cassette tape.

- You can check all programs stored by the six formats of SAVE command (see SAVE) by using VERIFY ⏎ or VERIFY "file name" ⏎ . In other words, it is not necessary to specify a format of SAVE command.
- If SAVE was not correctly performed, RW error is displayed. If this occurs, store the programs again.

Usage

The actual procedure is as follows.

Step 1: Tape Rewind — Return the tape on which programs are stored by the SAVE command to the initial location by using the tape counter.

Step 2: VERIFY Command Input — If the program has a file name, enter VERIFY "file name" ⏎ , and if it has no file name, enter VERIFY ⏎ .

Step 3: Press the PLAY button of the cassette tape recorder. When check starts, either PF B, AF B, PF A or DF A (see page 161) is displayed. If a program or data was correctly stored, Ready P0—P9 is displayed and the cassette tape stops. If the program or data was not correctly stored, RW error is displayed and the cassette tape stops.

4-2 PROGRAM COMMANDS

ANGLE

Function	Specifies the angle unit.
Format	ANGLE numerical expression (0≦numerical expression < 3)

The angle unit usually used for such as 30° and 60° is called DEGREE. However, RADIAN and GRADIENT are also used in mathematics. The PB-700 can provide a correspondence with all of them.

The ANGLE command is used to specify the following three angle units.
(1) DEGREE . . . . . (Example) 45°, 90° Input range of x: $-5400° < x < 5400°$
(2) RADIAN . . . . . (Example) 0.5π, 2π Input range of x: $-30\pi < x < 30\pi$
(3) GRADIENT . . (Example) 300, 1000 Input range of x: $-6000 < x < 6000$

The relationship between these angle units is as follows.

360 DEG (= 360°) = 2π RAD = 400 GRA

The angle unit is specified by the ANGLE command as follows.

ANGLE 0 — Specifies DEGREE
ANGLE 1 — Specifies RADIAN
ANGLE 2 — Specifies GRANDIENT.

It is set to ANGLE 0 (DEGREE) just after the power is turned on.

SAMPLE PROGRAM

```
10 REM *** ANGLE ***
20 ANGLE 0:PRINT SIN30;
30 ANGLE 1:PRINT SIN(PI/6);
40 ANGLE 2:PRINT SIN(100/3)
50 END
```

This program displays the value of SIN by using 3 angle units. All the results are 0.5.

BEEP

Function	Generates a buzzer sound.
Formats	BEEP BEEP 0 BEEP 1

BEEP command is provided in the PB-700 to generate a buzzer sound. There are many ways to use a buzzer sound. For example, when a long period of time is required for the execution of a program, execution termination is indicated by the sounding of the buzzer accomplished by inserting a BEEP command at the position of execution termination. Also, the fun of a game is increased by using this command.

The BEEP command has the following three formats.

(1) **BEEP**
Generates a relatively low buzzer sound.

(2) **BEEP 0**
Generates the same sound as a BEEP.

(3) **BEEP 1**
Generates a slightly higher buzzer sound.

SAMPLE PROGRAM

```
100 REM*** BEEP ***
110 IF INKEY$="" THEN 110
120 IF INKEY$="0" THEN BEEP 0
130 IF INKEY$="1" THEN BEEP 1
140 GOTO 110
```

This program was prepared to generate low buzzer sounds when 0 is pressed and high buzzer sounds when 1 is pressed.

CHAIN

Function	Loads a specified program and executes it from the first line.
Formats	CHAIN CHAIN "file name"

Explanation

When a CHAIN command appears during the execution of a program, program execution is stopped at that point, then a program with a file name that was specified by the CHAIN command is loaded from a micro cassette tape, and execution is performed from the beginning of the program.
If there is no file name, the program found first which is stored by SAVE or SAVE "file name" is loaded.
The CHAIN command has the following two formats.

(1) CHAIN
Loads PF B found first (program stored by SAVE or SAVE "file name") and executes it.

(2) CHAIN "file name"
Loads PF B of specified file name and executes it.

- Since the program is loaded in the presently specified program area, the previous program is erased with NEW.
- Programs stored by "SAVE ALL" and "SAVE, A" cannot be read in with the CHAIN command.
- If a password is attached to a loaded program, the password is also read in.
- Even when CHAIN is executed, variables are not cleared.

Usage

Input Lists 1—3 into program areas P1—P3, then store them on a cassette tape.

⟨List 1⟩. . . . . Program that computes the area of a circle.
⟨List 2⟩. . . . . Program that computes the area of a triangle.
⟨List 3⟩. . . . . Program that computes the area of a square.

Also, input List 0 into P0 and execute it. List 0 is used to select List 1 — List 3 by the CHAIN command on lines 70—90 to read it into the PB-700, and to execute the computation.

List 0

```
  10 REM CHAIN PRO.0
  20 CLS :PRINT "AREA CALCULATIONS"
  30 PRINT "1"+CHR$(180)+CHR$(221)+"   2
"+CHR$(187)+CHR$(221)+CHR$(182)+CHR$(184
);
  40 PRINT "   3"+CHR$(188)+CHR$(182)+CH
R$(184)
  50 PRINT "SELECT NO.    "
  60 BB$=INKEY$:IF VAL(BB$)>3 THEN 60 E
LSE IF VAL(BB$)<1 THEN 60
  70 IF BB$="1" THEN CHAIN "PRO.1"
  80 IF BB$="2" THEN CHAIN "PRO.2"
  90 CHAIN "PRO.3"
```

List 1

```
10 REM PRO.1
20 PRINT "RADIUS";
30 INPUT RR
40 S=PI*RR^2
50 PRINT S
60 END
```

List 2

```
10 REM PRO.2
20 PRINT "HEIGHT";
30 INPUT HH
40 PRINT "BASE";
50 INPUT LL
60 S=HH*LL/2
70 PRINT S
80 END
```

List 3

```
10 REM PRO.3
20 PRINT "LENGTH";
30 INPUT HH
40 PRINT "WIDTH";
50 INPUT LL
60 S=HH*LL
70 PRINT S
80 END
```

CLEAR

Function	Clears all variables.
Format	CLEAR

Explanation

The CLEAR command is used to clear all numerical variables and character variables.
Numerical variables are cleared to 0 and character variables are cleared to " " (null-string). Also, at the same time, registered variables in a program are deleted, and the defined array variables are deleted.
Since a FOR nesting stack is cleared, a FOR-NEXT loop cannot be continued.

Usage

The program below is used to display data by totalizing the sum of data and the number of data. However, if the program is executed again after pressing BRK key, the correct answer cannot be obtained since the numerical values are assigned as they are to variables S and N at the first execution. Therefore, the CLEAR command was inserted between line 10 and line 30 as shown in the program on the right so that a correct answer can be obtained every time when execution is performed many times.

```
10 PRINT "TOTAL"
30 INPUT "D=",D
40 S=S+D
50 N=N+1
60 PRINT "S(";N;")=";S
70 GOTO 30
```

```
10 PRINT "TOTAL"
20 CLEAR
30 INPUT "D=";D
40 S=S+D
50 N=N+1
60 PRINT "S(";N;")=";S
70 GOTO 30
```

CLS

Function	Clears all displays and moves the cursor to the home position (top left corner).
Format	CLS

Explanation

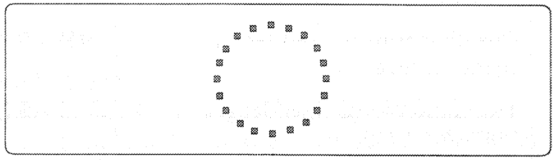
CLS, an abbreviation for clear screen, is used to clear the screen and to move the cursor to the home position at the top left corner.
It is used to clear the screen once during a graphic display.

(1) When this command is manually executed, the cursor is displayed at the (0,1) position.
(2) The cursor is displayed at the (0,0) position during program execution.

Usage

```
10 CLS
20 FOR I=0 TO 360 STEP 12
30 DRAW(SINI*15+80,COSI*15+15)
40 NEXT I
50 END
```

This program is used to display the following pattern.



When a pattern is drawn on the screen as shown above, the screen must be cleared before drawing starts. Therefore, the CLS command is used at the beginning of a program.

DIM

Function	Declares an array.
Format	DIM array variable name (subscript) [, array variable name (subscript)]

The DIM command declares an array by the specified variable name in the memory area. Variables declared by DIM (called array variables) include single-precision numerical arrays, half-precision numerical arrays, fixed-length character arrays in which up to 16 characters can be stored, and defined-length character arrays in which characters from 0 to 79 can be freely defined.

The general format of DIM can be indicated as follows.

DIM array variable name (subscript [, subscript]) [, array variable name (. . . . . . .
(Maximum value of subscript : 255)

Examples of declaration statements for various arrays are provided as follows.

Format	Classification	Example
DIM array variable name (I)	One-dimensional single-precision numerical array	DIM A (5)
DIM array variable name (I, J)	Two-dimensional single-precision numerical array	DIM A (2, 3)
DIM array variable name ! (I)	One-dimensional half-precision numerical array	DIM A! (5)
DIM array variable name ! (I, J)	Two-dimensional half-precision numerical array	DIM A! (2, 3)
DIM array variable name$ (I)	One-dimensional fixed-length character array	DIM A$ (5)
DIM array variable name$ (I, J)	Two-dimensional fixed-length character array	DIM A$ (2, 3)
DIM array variable name$ (I)*N	One-dimensional defined-length character array	DIM A$ (5)*20
DIM array variable name$ (I, J)*N *N can be omitted	Two-dimensional defined-length character array	DIM A$ (2, 3)*20

I, J and N are real numbers or numerical expressions with the range of $0 \leqq I < 256$, $0 \leqq J < 256$ and $0 \leqq N < 80$, in which the fractional part of numerical value is discarded.
An array variable name is one character from among the capital alphabetical characters from A to Z.
The maximum dimensional value is 2, which allows one-dimensional arrays and two-dimensional arrays to be specified.
A half-precision numerical array can be specified by placing " ! " just after the array variable name, and a character array can be specified by placing " $ " just after the array variable name.
A character array, in which a character string with a "N" length can be assigned, is declared by placing "*N". However, if "*N" is omitted, it is a fixed-length character array with N = 16, i.e. an array of 16 characters.

Usage

Enter the following program and run it.

```
 10 CLEAR
 20 DIM A(2,3), B(2,3)
100 END
```

An operational expression may or may not be written between line 20 and line 100.
When characters such as Ready P0 appear after program execution, check the array variable list, then it is displayed as follows.

```
LIST   V ⏎
A( )   B( )
Ready  P0
```

The array variable name declared by DIM can be confirmed by using

```
LIST  V ⏎
```

as mentioned above.
Let's check the content of each array variable by adding the following list to the program mentioned above. An array variable must be declared by DIM before it is used.

```
30  FOR I =0 TO 2
40  FOR I =0 TO 3
50  PRINT A( I , J );B( I , J );
60  NEXT J : NEXT I
```

When you run the program after adding lines 30 to 60, the following is displayed.

■ Execution Example

```
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
0  0  0  0
Ready  P0
```

Here, the content of 24 array variables, A (0, 0), B (0, 0)—A (2, 3), B (2, 3), is displayed as "0".
It is important that the content of all the arrays becomes "0" when the DIM command is executed.
While the content of the numerical arrays mentioned above is 0, when character arrays are declared, their content becomes a null-string in which nothing is displayed. Null-string means that a character string is empty.
Precautions shall be taken on the difference between space-strings and null-strings. A space-string is a string that has one space (A$ (I) = "␣") and a null-string is a string that is empty (A$ (I) = " ").

SAMPLE PROGRAM 1

* Rearrange (sort) program (one-dimensional numerical array).

```
10 CLEAR
20 DIM D(5)
30 FOR I=1 TO 5
40 PRINT "DATA";I;" =";:INPUT D(I)
50 NEXT I
60 REM SORT
70 F=0
80 FOR I=1 TO 4
```

```
 90 IF D(I)<D(I+1) THEN X=D(I):D(I)=D(
I+1):D(I+1)=X:F=1
 100 NEXT I
 110 IF F=1 THEN 70
 120 REM RESULT
 130 FOR I=1 TO 5
 140 PRINT D(I);
 150 NEXT I
 160 END
```

This program enters 5 numerical data and sorts these data in a sequence with the largest value first.
D(0) — D(5) utilization is declared by the DIM command on line 20, however five arrays (D (1)—D (5)) are used here.
Lines 60 to 110 contain the sort program while lines 120 to 150 contain the program which displays the result with sorting performed in a sequence with the largest value first.
It is convenient to use array variables by combining them with the FOR-NEXT command as shown in the sample program.

SAMPLE PROGRAM 2

* Vertical and horizontal totalization (Two-dimensional numerical array)

```
10 DIM A(3,3),X(3),Y(3)
20 FOR I=1 TO 3
30 FOR J=1 TO 3
40 PRINT "(";I;",";J;")";"="
50 INPUT A(I,J)
60 NEXT J:NEXT I
70 REM SUBTOTAL
```

```
 80 FOR I=1 TO 3
 90 FOR J=1 TO 3
100 X(I)=X(I)+A(I;J)
110 Y(J)=Y(J)+A(I,J)
120 NEXT J:NEXT I
130 REM RESULT
140 FOR I=1 TO 3
150 PRINT "X(";I;")=";X(I)
160 PRINT "Y(";I;")=";Y(I)
170 FOR K=1 TO 100:NEXT K
180 NEXT I
190 END
```

This is a program that assigns the data in Table 1 to a two-dimensional array as shown in Table 2 in order to obtain vertical and horizontal subtotals X(1), X(2), X(3), and Y(1), Y(2), Y(3).

Table 1

14	9	21	X (1)
35	4	53	X (2)
6	15	11	X (3)
Y (1)	Y (2)	Y (3)	

Table 2

A (1, 1)	A (1, 2)	A (1, 3)	X (1)
A (2, 1)	A (2, 2)	A (2, 3)	X (2)
A (3, 1)	A (3, 2)	A (3, 3)	X (3)
Y (1)	Y (2)	Y (3)	

Lines 80 to 120 provide the program that obtains a subtotal while lines 130 to 180 provide the program that displays the obtained values.

SEE CLEAR, NEW ALL, ERASE, LIST V

DRAW/DRAWC

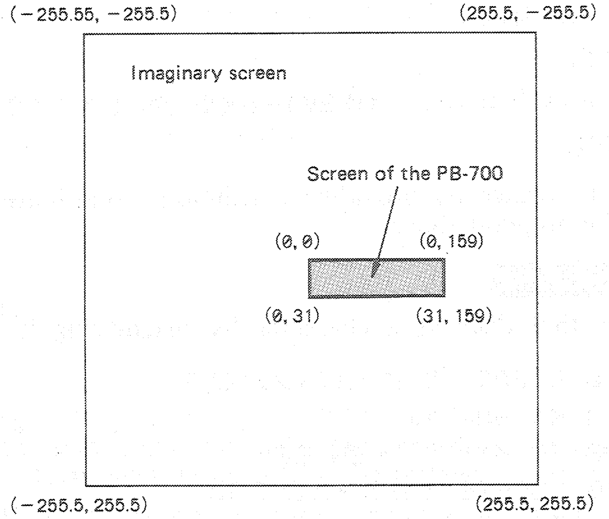
Functions	DRAW: Draws a dot. DRAWC: Clears a dot.
Formats	DRAW (X_1, Y_1) $[-(X_2, Y_2)]$ DRAWC (X_1, Y_1) $[-(X_2, Y_2)]$

The DRAW command draws a dot or a line on the screen while DRAWC clears them.
Since not only characters but dots and lines can be displayed on the screen, many kinds of graphs, etc. can be made with this command.

Coordinates that can be specified by DRAW (X, Y) or DRAWC (X, Y) are as follows.

$-255.5 < X < 255.5$ $-255.5 < Y < 255.5$

Since the range of the screen dot coordinates are $0 \leqq X \leqq 159$, and $0 \leqq Y \leqq 31$, the imaginary screen shown below can be considered.



In this figure, ▨ is equivalent to the screen.
Therefore, all of the virtual screen can be displayed by changing the screen relative location.
However, the top left corner of the screen is the origin (0, 0).

Usage

DRAW and DRAWC are used as follows.

DRAW (X_1, Y_1)	Draws a dot at (X_1, Y_1).
DRAWC (X_1, Y_1)	Erases the dot at (X_1, Y_1).
DRAW (X_1, Y_1)–(X_2, Y_2)	Draws a line from (X_1, Y_1) to (X_2, Y_2).
DRAWC (X_1, Y_1)–(X_2, Y_2)	Erases the line from (X_1, Y_1) to (X_2, Y_2).

X and Y mentioned above are numerical values, variable names, and numerical expressions with the following range.

$$-255.5 < X < 255.5 \qquad -255.5 < Y < 255.5$$

They are shown on the actual screen with integer values which are rounded at one decimal place.
The following program draws a rectangle on the screen.

```
10 REM DRAW
20 CLS
30 DRAW(10,10)-(10,20)-(150,20)-(150,10)-(10,10)
40 END
```

A figure can be drawn by providing continuous coordinates with "–" as shown in the program above.

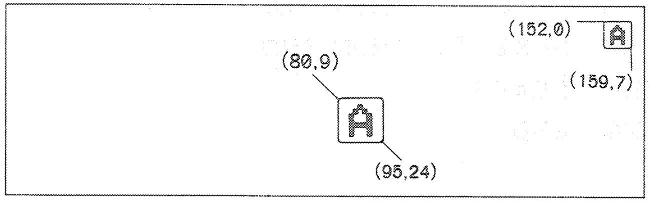
SAMPLE PROGRAM

* A program that displays a character by magnifying it two times.

```
10 CLEAR :DIM A(7,7):CLS
20 K$=INKEY$
30 IF K$="" THEN 20
40 LOCATE 19,0:PRINT K$
50 FOR I=0 TO 7
60 FOR J=0 TO 7
70 A(I,J)=POINT(I+152,J)
80 NEXT J:NEXT I
90 FOR I=0 TO 7
```

```
100 FOR J=0 TO 7
110 IF A(I,J)<>1 THEN 160
120 DRAW(2*I+80,2*J+9)
130 DRAW(2*I+81,2*J+9)
140 DRAW(2*I+80,2*J+10)
150 DRAW(2*I+81,2*J+10)
160 NEXT J:NEXT I
170 LOCATE 0,1
180 END
```

This program uses INKEY$ to enter one character, and first displays the character at the position which is specified by LOCATE (19,0) as shown in the figure below.
This character is displayed by using dots in a square area with a diagonal line formed by (152,0)–(159,7) in which the dots that are lit and turned off are checked with the POINT function, then values 0 and 1 are assigned to the array variable A (I, J).
By using this array variable as data, the character that is magnified two times is displayed in a square area with a diagonal line formed by (80,9)–(95,24).



SEE POINT

Note:

When drawing a straight line by the DRAW command, a dot located on the edge of the screen may not be drawn according to calculation error. In this case, please manage by either the following method (1) or (2).

(1) Draw only the corresponding dot on the edge of the screen.
(2) To correct the calculation error, specify the ending dot of the straight line as the edge of the screen (X_2 = 159 or Y_2 = 31), and set the ending dot of the straight line as follows.

DRAW (X_1, Y_1) – (X_2, Y_2 + 0)
X_2 = 159 or Y_2 = 31

END

Function	Terminates the execution of a program.
Format	END

The END command terminates the execution of a program. A nested stack (control of FOR-NEXT loop and GOSUB) is cleared by the execution of an END command. As many END statements as desired can be placed anywhere in a program.
The END command placed at the end of a program can be omitted.

SAMPLE PROGRAM

```
500  FOR 1=0 TO 1000
510  K$=INKEY$
520  IF K$="A" THEN 1000
530  IF K$="B" THEN 2000
540  IF K$="C" THEN 3000
550  IF K$="D" THEN END
560  NEXT I
570  END
```

This program is used as part of a menu program.
After the entry of characters such as "A", "B", and "C" is performed, each one jumps to the program on line 1000, line 2000, and line 3000, respectively, for the execution of these programs.
When "D" is entered the program execution is immediately terminated. However, when key entry is performed with a key other than "A"~"D", or when no key entry is performed, it is automatically terminated after a certain period of time.

SEE STOP

ERASE

Function	Releases registered variables and array variables with a variable name unit.
Format	ERASE variable name [, variable name]

An ERASE command can release registered variables and array variables that can be confirmed by LIST V with a variable name unit.
Specification of a variable name for release is performed as follows.

Variable names
displayed by a LIST V: AB, AB$, A ( ), A! ( ), A$ ( )

ERASE AB, AB$, A, A!, A$

The nesting stack is cleared by execution of the ERASE command.
When an unregistered variable name is specified, it proceeds to the following execution without any operation.

SEE DIM, LIST V

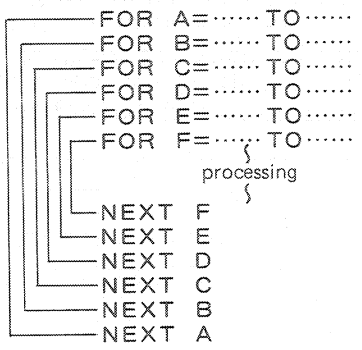
FOR~TO~STEP/NEXT

Function	Repeats the execution from FOR to NEXT the specified number of times.
Formats	FOR numerical variable = *i* TO *j* [STEP *k*] ~ NEXT numerical variable (same variable as that of the FOR statement) *i* : Initial value of the variable. *j* : Final value of the variable. *k*: Increment

The FOR-NEXT command executes each statement between the FOR command and the NEXT command repeatedly by the specified number of times.

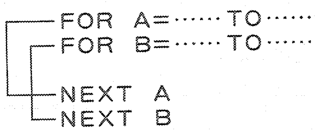
This command has some restrictions as follows.

(1) Variables must be numerical variables.
(2) STEP *k* can be omitted. In this case +1(STEP 1) is set as the increment.
(3) A variable and a numerical expression can be used for the initial value *i*, final value *j*, and increment *k*.
(4) If $k > 0$ and $i > j$, FOR-NEXT execution is performed only once.
(5) If $k < 0$ and $i < j$, FOR-NEXT execution is performed only once.
(6) When a FOR-NEXT execution is performed and the execution is moved to the following line, a control variable becomes a value of $i + nk$ which is larger than *j*.
(7) Nesting of FOR-NEXT loop can be performed with up to 6 levels.



```
FOR  A=······ TO······
FOR  B=······ TO······
FOR  C=······ TO······
FOR  D=······ TO······
FOR  E=······ TO······
FOR  F=······ TO······
              ~
          processing
              ~
NEXT  F
NEXT  E
NEXT  D
NEXT  C
NEXT  B
NEXT  A
```

(8) Nesting of FOR-NEXT loop must be perfect. If the interval is crossed, an error (FO error) occurs.



```
FOR  A=······TO······
FOR  B=······TO······

NEXT  A
NEXT  B
```

(9) As many NEXT as desired can exist for one FOR.
(10) When FOR is not executed and NEXT appears, an error (FO error) occurs.
(11) The variable for NEXT command cannot be omitted.
(12) If ERASE or CLEAR is executed, an FO error occurs in the following NEXT statement because the FOR-NEXT stack is cleared.
(13) When an identical control variable is used, a new loop becomes effective. However, all loops inside the new loop are cleared.
(14) A jump into a FOR-NEXT statement can be performed, however a jump into a loop using GOTO, GOSUB statement, etc. cannot be performed.

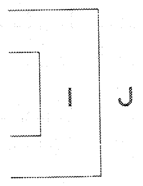
Usage

The following program is used to demonstrate the change of the variable, depending on the value of the initial value, final value, and increment.

```
10  INPUT "FOR I=",I,"TO",J,"STEP",K
20  PRINT "FOR I=";I;"TO";J;"STEP";K
30  FOR A=I TO J STEP K
40  PRINT "VARIABLE:A=";A
50  FOR X=1 TO 100:NEXT X
60  NEXT A
70  PRINT :GOTO 10
```

Several execution examples are provided below.
In the following program, the FOR-NEXT loop has two levels.

```
10  CLEAR
20  DIM A(9,9)
30  FOR I=1 TO 9
40  FOR J=1 TO 9
50  A(I,J)=I*J
60  NEXT J
70  NEXT I
```



In this example, multiplication is performed and operation result is assigned to array variable A(I, J) between the two FOR-NEXT loops. Since execution is performed 9 times by the internal FOR-NEXT command and execution is performed 9 times by the external FOR-NEXT command, a total of 9 x 9 operations are performed.
In regard to the two level FOR-NEXT command in this program, the internal FOR-NEXT and the external FOR-NEXT must correspond with each other.
This also applies to the levels from 3 to 6.

The following program changes the operation by jumping from the loop depending on the arithmetic result of the FOR-NEXT statement.

```
 10  CLEAR
 20  FOR I=1 TO 100
 30  X=X+I
 40  IF X>=1000 THEN 100
 50  NEXT I
 60  END
100  PRINT I;X
110  END
```

This program performs the addition of 1 + 2 + 3 . . . . . , and when the result exceeds 1000, it jumps to line 100 and displays the value of the variable at that time and the addition result.

A technique to jump out from a loop in the middle by using an IF-THEN command is often used.
A technique for jumping out from a loop once by using a GOSUB command and returning to continue execution can also be used.

The following program provides an example of when there is nothing to execute between FOR-NEXT.

```
10  LOCATE 8,2 : PRINT "HIT!";
20  FOR I=0 TO 50
30  NEXT I
40  CLS
50  FOR I=0 TO 50
60  NEXT I
70  GOTO 10
```

There is no statement to be processed between the FOR-NEXT on lines 20–30 and lines 50–60. However, the FOR-NEXT command is executed the specified number of times even in this case.
This command which seems to be nonsense is often used to kill time, and is called a "waiting loop".
In this program, the characters "HIT!" are displayed for a certain period of time, erased for a certain period of time, and displayed again for a certain period of time repeatedly.
As the final value of the variable becomes large, the waiting time becomes longer.

SAMPLE PROGRAM

* A program that provides a display by increasing "*" by one each time.

```
10 CLS
20 N=1
30 FOR I=1 TO N
40 PRINT "*";
50 NEXT I
60 PRINT
70 N=N+1
80 IF N>=20 THEN END  ELSE 30
```

This program increases the final value of the FOR-NEXT command by one each time and increases the number of "*" displayed by one each time. The execution example is as follows.

```
*
**
***
****
*****
******
```

SEE IF~THEN~ELSE

GET

Function	Reads data stored on a cassette tape to a variable.
Formats	GET variable [, variable] GET "file name" variable [, variable]

The GET command is used to read data stored on a cassette tape by the PUT command to a variable.
Although the file name can be omitted as shown in the above formats, the file that appears first on cassette tape (MT) is read. The data that is read can be sequentially assigned to a different variable by punctuating a variable with a comma (,).
However, when data is assigned to a numerical variable, the space at the beginning of data is ignored.

Usage

The following program stores the values of variables A, B, C and D with the file name "TEST" on a cassette tape.

```
10  REM PUT MT
20  A=10:B=20:C=30:D=40
30  PUT "TEST" A, B, C, D
40  END
```

The following example program reads data from the "TEST" file on a cassette tape by using a GET command.

```
10  REM GET MT
20  GET "TEST" E, F, G, H
30  PRINT E;F;G;H
40  END
```

In this example, the data of A, B, C, and D are read into the variables E, F, G, and H. An important item is that data is stored on cassette tape in an A, B, C, D sequence.
Although data is read in an A, B, C, D sequence during cassette tape playback, the data of A is assigned to E, and data of B is assigned to F, ... sequentially.
If GET is executed with variables that exceed the variable data stored on a cassette tape, an error (DA error) occurs because of data shortage.

SAMPLE PROGRAM

* A program to store a person's name and data on cassette tape, and to read and display a person's data when a name is entered.

```
 10 CLEAR
 20 DIM N$(10),D$(10)
 30 INPUT "PUT:O GET:I END:E";M$
 40 IF M$="O" THEN GOSUB 100
 50 IF M$="I" THEN GOSUB 300
 60 IF M$="E" THEN END
 70 GOTO 30
100 I=0
110 INPUT "NAME:";N$(I)
120 PUT N$(I)
130 IF N$(I)="0" THEN 180
140 INPUT "DATA:";D$(I)
150 PUT D$(I)
160 I=I+1
170 IF I<10 THEN 110
180 RETURN
300 J=0
310 INPUT "NAME:";A$
```

```
  320 GET N$(J)
  330 IF N$(J)="0" THEN PRINT "NOT FOUND
":GOTO 380
  340 GET D$(J)
  350 IF A$<>N$(J) THEN 370
  360 PRINT A$;"=";D$(J):GOTO 380
  370 J=J+1:GOTO 320
  380 RETURN
```

This program consists of two subroutines.
Lines 100—180 provide a subroutine to store data on a cassette tape, while lines 300—380 provide a subroutine to read data from a cassette tape, and to search data.
A name is entered in array variable N$(I), and his data is entered in D$(I). Then storing and reading are performed with these two variables as a pair.
When "O" is entered for the menu on line 30, the storing subroutine on line 100 and after is executed with the input of a name and data continuously performed. Up to 11 data inputs can be performed. However, since it is necessary to attach "0" to the end of the last name (to indicate data end), up to 10 data inputs can actually be performed.
When "I" is entered for the menu on line 30 to search for the name of a person, then the name and data are read from a cassette tape.
After the name is found, the name and data are displayed and the program returns to the menu on line 30.
Enter "E" for the menu to terminate the program.

SEE PUT, SAVE

GOSUB/RETURN

Function	Causes a jump to a subroutine and a return to a main program.	
Formats	GOSUB line number GOSUB PROG *n* RETURN	1 ≦ line number < 10000 0≦*n*<10 (*n* indicates the program area No.)

The GOSUB command causes a jump to a subroutine of a line which is specified by a variable or a numerical expression, and the RETURN command causes a return from the subroutine to the main program.

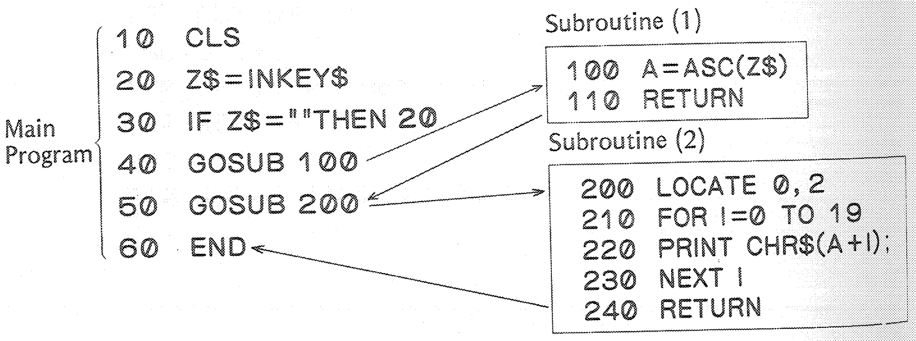
Example

	Line number	Program area
Numerical constant	GOSUB 200	GOSUB PROG 4
Numerical variable	GOSUB N	GOSUB PROG L
Numerical expression	GOSUB N ∗5	GOSUB PROG L/10

The PB-700 allows a jump to a subroutine and a return to a main program in other program areas, as well as a jump to a subroutine in the same program area as shown by the above formats.

Usage

In the following program, the execution flow is shown by changing the layout.



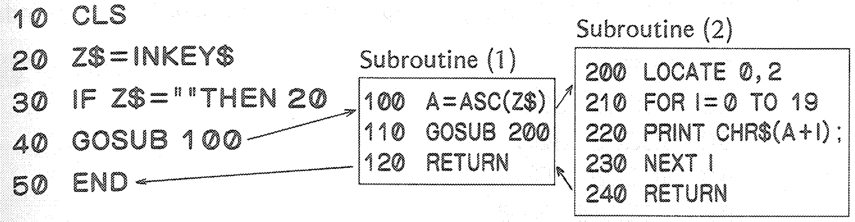
```
10  CLS
20  Z$=INKEY$
30  IF Z$=""THEN 20
40  GOSUB 100
50  GOSUB 200
60  END
```

```
100  A=ASC(Z$)
110  RETURN
```

```
200  LOCATE 0,2
210  FOR I=0 TO 19
220  PRINT CHR$(A+I);
230  NEXT I
240  RETURN
```

Lines 10–60 provide the main program, lines 100–110 provide one subroutine, while lines 200–240 provide another subroutine.
When the RETURN command is executed as shown by the arrows, a return is made to the statement next to the GOSUB command and the execution will continue.

This program can be rewritten as follows by changing its format slightly.



```
10  CLS
20  Z$=INKEY$
30  IF Z$=""THEN 20
40  GOSUB 100
50  END
```

```
100  A=ASC(Z$)
110  GOSUB 200
120  RETURN
```

```
200  LOCATE 0,2
210  FOR I=0 TO 19
220  PRINT CHR$(A+I);
230  NEXT I
240  RETURN
```

Although the content of this program is exactly the same as the previously mentioned program, this program has a double structure in which subroutine (2) is included in subroutine (1).
Up to 12 nesting levels can be performed with the GOSUB/RETURN command.
Precautions shall be taken concerning the following items when a GOSUB/RETURN command is used.

(1) One subroutine must have one RETURN command, and can have as many as desired.
(2) If there is no specified line when a GOSUB command is executed, an error (UL error) will occur.
(3) When GOSUB PROG *n* is executed, if a program is not written in program area *n*, an error (UL error) will occur in the present program area.
(4) When there is no GOSUB command for a RETURN command, an error (GS error) will occur.

(5) Up to 12 GOSUB nesting levels can be performed. (If there are 13 levels or more, an error (NO error) will occur.)
(6) When a fraction is included in a line with a line number specified by GOSUB, or in a program area specified by GOSUB, execution is performed with the fraction discarded.

SAMPLE PROGRAM

* A slot machine game program.

```
 10 CLS
 20 GOSUB 200
 30 GOSUB 300
 40 GOSUB 400
 50 LOCATE 3,0
 60 PRINT "POINT";N
 70 LOCATE 0,0:END
200 X=INT(10*RND)
210 Y=INT(10*RND)
220 Z=INT(10*RND)
230 RETURN
300 LOCATE 6,2
310 PRINT X;"   ";Y;"   ";Z;
320 RETURN
400 IF X=Y THEN IF Y=Z THEN N=100:RETU
RN
410 IF X=Y THEN N=40:RETURN
420 IF Y=Z THEN N=30:RETURN
430 IF X=Z THEN N=20:RETURN
440 N=0
450 RETURN
```

This program displays 3 numerals. If all of the 3 numerals are identical, 100 points are given, and if 2 numerals are identical, 40, 30 or 20 points are given depending on their locations.
If all 3 numerals are different, 0 points are displayed.
Lines 200—230 provide the subroutine that generates the 3 numerals with the RND function, while lines 300—320 provide a subroutine that displays the numerals at the center of the screen, and lines 400—450 provide a subroutine that checks the number of points given.
Lines 10—70 provide the main program which simply controls each subroutine and displays the number of points given.

SEE GOTO

GOTO

Function	Causes an unconditional jump to a specified line.	
Formats	GOTO line number	1 ≦ line number < 10000
	GOTO PROG *n*	0 ≦ *n* < 10 *n* : Indicates the program area No.

The GOTO command unconditionally jumps to a specified line number or to the beginning of a program area which is specified by a variable or a numerical expression.
If there is no specified line number, or no program in the specified program area, an error (UL error) will occur.
If a fraction is included in a specified line or in a specified program area, the fraction is discarded when execution is performed.

Example

	Line number	Program area
Numerical constant	GOTO 500	GOTO PROG 4
Numerical variable	GOTO N	GOTO PROG N
Numerical expression	GOTO N*5	GOTO PROG N/10

SAMPLE PROGRAM

* Simplified Digital Clock Program.

```
    10 INPUT "H=";H,"M=";M,"S=";S
    20 CLS
    30 IF H<10 THEN LOCATE 8,0:PRINT H;":
";:GOTO 50
    40 LOCATE 7,0:PRINT H;":";
    50 IF M<10 THEN LOCATE 12,0:PRINT M;"
:";:GOTO 70
    60 LOCATE 11,0:PRINT M;":";
    70 IF S<10 THEN LOCATE 16,0:PRINT S;"
:";:GOTO 90
    80 LOCATE 15,0:PRINT S;
    90 FOR I=2 TO 120:NEXT I
   100 S=S+1
   110 IF S>=60 THEN 200
   120 GOTO 70
   200 S=0:M=M+1
   210 IF M>=60 THEN 300
   220 GOTO 50
   300 M=0:H=H+1
   310 IF H>=24 THEN H=0
   320 GOTO 30
```

When you run this program, H (hour), M (minutes), and S (seconds) are requested. After the present time is entered and the ↵ key is pressed, the time is displayed with second unit.
Since the internal clock of the microcomputer is not used for this program, it does not show the exact time, but indicates the approximate time.
If it gains, adjust it by increasing the final value of the FOR command on line 90, and if it loses, adjust it by decreasing the final value.
Many GOTO commands are used in this program.
The program creates an infinite loop by using GOTO commands on lines 120, 220, and 320.

SEE GOSUB/RETURN, IF-THEN-ELSE

IF~THEN~ELSE

Function	Executes the contents after THEN or ELSE depending on the condition after IF.
Formats	IF conditional expression THEN {line number / command} ELSE {line number / command} A numerical expression cannot be used as a line number.

The IF-THEN command performs a conditional jump while a GOTO command performs an unconditional jump.
When the conditional expression is true, the THEN statement is executed, and when it is false, the ELSE statement is executed. If there is not an ELSE statement, the following line is executed.
A line number, or a command statement to which a jump is performed can be inserted in THEN and ELSE statements.
Multi-statements can be performed in THEN or ELSE statements by using a colon (:) as shown below.

$$\text{IF conditional expression }\underbrace{\text{THEN} \sim : \sim : \sim}_{\text{THEN statement}}\underbrace{\text{ELSE} \sim : \sim : \sim}_{\text{ELSE statement}}$$

An IF statement can be inserted in the THEN or ELSE statement mentioned above as shown below.

IF conditional expression (1) THEN IF conditional expression (2) THEN ~ ELSE ~ ELSE ~

Executes when conditional expressions (1) and (2) are both true.

Executes when conditional expression (1) is true and when conditional expression (2) is false.

Executes when conditional expression (1) is false.

Usage

Examples of IF ~ THEN ~ ELSE statement usage are shown below. They check whether a triangle can be made or not after three sides are entered.

(1)

```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN 40
30 GOTO 50
40 IF ABS(A-B)<C THEN 60
50 PRINT "NOT TRI" : GOTO 10
60 PRINT "TRI" : GOTO 10
```

(2)

```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN IF ABS(A-B)<C THEN 40
30 PRINT "NOT TRI" : GOTO 10
40 PRINT "TRI" : GOTO 10
```

(3)

```
10 INPUT "A=";A, "B=";B, "C=";C
20 IF A+B>C THEN IF ABS(A-B)<C THEN 40
30 PRINT "NOT TRI" : GOTO 10
40 IF A=B THEN IF B=C THEN PRINT "E.TRI" : GOT
   O 10 ELSE PRINT "I.TRI" : GOTO 10
50 IF B=C THEN PRINT "I.TRI" : GOTO 10
60 IF A=C THEN PRINT "I.TRI" : GOTO 10 ELSE
   PRINT "TRI" : GOTO 10
```

Example programs (1) and (2) check whether that is a triangle (TRIangle) or not a triangle (NOT TRIangle) by having the three sides of a triangle, A, B, and C, entered.
In example(1), two triangle conditions (A+B>C, |A−B|<C) are checked on separate lines (line 20 and line 40), while in example (2), they are checked on one line (line 20).
Also, in example(3), the program checks whether that is an equilateral triangle (Equilateral TRIangle) or an isosceles triangle (Isosceles TRIangle), as well as whether that is a triangle (TRI) or not a triangle (NOT TRI).
In example (3), whether that is a triangle (TRI) or not is checked first on line 20. If there is not a triangle, "NOT TRI" is displayed and a return is made to line 10.
If there is a triangle, whether the three sides are equal or not is checked on line 40. If the three sides are equal, "E.TRI" is displayed. If two sides are equal, "I.TRI" is displayed, and if all three sides are not equal, "TRI" is simply displayed on lines 40 to 60, and a return is made to line 10.

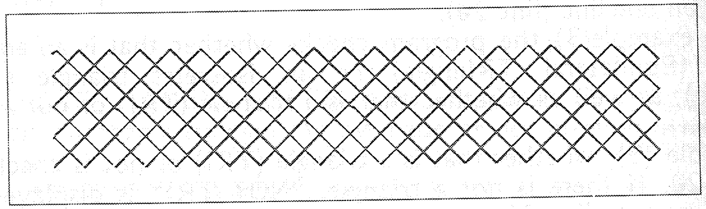
SAMPLE PROGRAM

* A program to draw a pattern on the screen with dots.

```
 10 CLS
 20 X=0:Y=0:N=1:M=1
 30 DRAW(X,Y)
 40 X=X+N:Y=Y+M
 50 IF X>=159 THEN N=-1
 60 IF Y>=31 THEN M=-1
 70 IF X<=0 THEN N=1
 80 IF Y<=0 THEN M=1
 90 IF X=1 THEN IF Y=31 THEN BEEP 1:END
100 GOTO 30
```

This program determines whether or not the value of the dot coordinates (X, Y) on the screen is in the screen limitation by the IF-THEN command (lines 40–80) to control whether the values of X, Y coordinate are increased or decreased.
When the values of X and Y become X = 1, Y = 31 (line 90), a beep sound is generated, and the program is terminated.
The next figure shows the execution result.

■ **Execution Example**



INPUT

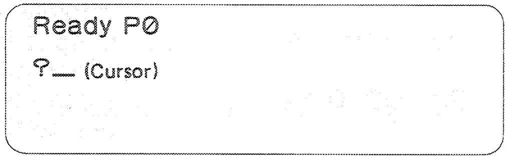
Function	Requests data entry (numerical value, character) from the keyboard to a variable.
Formats	INPUT variable [, variable] INPUT “prompt statement”, variable [, “prompt statement”, variable] INPUT “prompt statement”; variable [, “prompt statement”; variable]

The INPUT command is a representative input command used to enter data from the keyboard to a variable.
The basic formats of an INPUT statement are as follows.

Example 1	INPUT A
Example 2	INPUT X, Y, Z
Example 3	INPUT “AGE”, A
Example 4	INPUT “NAME”; A$
Example 5	INPUT “X=; X, “Y=”; Y

When the INPUT command is executed, the personal computer displays an input request message, and waits for data input.
For example, when example 1 is executed, “ ? ” is displayed as follows and the cursor turns on and off at the right of “ ? ”. The data input setup has been completed.

Display during INPUT command execution.



```
Ready P0
?_ (Cursor)
```

Data input is performed by pressing keys. Always press the ENTER key or ↵ key at the end of data input. Data input is performed by this operation. It should be noted that the ENTER key functions the same as the ↵ key during INPUT statement execution.

▪ **Variables that can be used in an INPUT statement are as follows.**

	[Examples]
Numerical variable	INPUT X
Character variable	INPUT X$ (Up to 7 characters can be entered.)
Registered variable	INPUT XY INPUT XY$ (Up to 16 characters can be entered.)
Array variable	INPUT X! (*i*), X! (*i*, *j*) Half-precision numerical array INPUT X (*i*), X (*i*, *j*) Single-precision numerical array INPUT A$ (*i*), INPUT A$ (*i*, *j*) Character array Fixed-length character array — Up to 16 characters can be entered. Defined-length character array — Up to 79 characters can be entered.

Usage

(1) INPUT statement in which the input of a numerical value is performed.

Let's check INPUT statement usage and functions by using a simple program.

10 INPUT A	Input instruction. Provides data input to variable A.
20 PRINT A	Output instruction. Displays the content of variable A.
30 GOTO 10	Jump instruction. Moves program execution to line 10.

After inputting this program, enter RUN ↵ or SHIFT $\overset{\text{PO}}{0}$ to execute it. " ? " is displayed. Now, enter 3.6 ↵. If the entry is performed correctly, the same numerals are displayed again by a PRINT statement as follows.

CLS SHIFT $\overset{\text{PO}}{0}$ 3.6 ↵

```
?3.6
 3.6
?_
```

Only when the input of a numerical value is performed by an INPUT statement, input can be performed by calculation expression, instead of numerical data.
Let's confirm this by using the previous program.

CLS SHIFT $\overset{\text{PO}}{0}$ 100-20/5 ↵

```
?100-20/5
 96
?
```

(2) INPUT statement in which character input is performed.

Perform character input by changing the program of (1). The program is as follows.

```
10 INPUT A$
20 PRINT A$
30 GOTO 10
```

When character input is performed, a character variable is used as shown above. When you execute this program, input is requested by the display of " ? " the same as for numerical value input. When entering the character string ABC, the display becomes as follows.

CLS SHIFT $\overset{\text{PO}}{0}$ ABC ↵

```
?ABC
ABC
?_
```

When the numerical value 123 is entered, 123 is displayed. However, it should be noted that this (123) is a character string and not a numerical value.
When data is input to a character variable, it is unnecessary to enclose the character string with " " (double quotation marks). If " " is used, it is also entered as character data.

■ Data that can be input to each variable by an INPUT statement.

A. Numerical variable
- a. $\pm 1 \times 10^{-99}$ to $\pm 9.99999999999 \times 10^{99}$ and 0
- b. Operational expression for a numerical value (Example: 200 x (5 + 2))
- c. Numerical variable from A to Z (Fixed variable)
- d. Registered variable
- e. Array variable

B. Character variable

a. Fixed character variable	Up to 7 characters and symbols.
b. Registered character variable	Up to 16 characters and symbols.
c. Array variable	
● Fixed-length character array	Up to 16 characters and symbols.
● Defined-length character array	Up to 79 characters and symbols.

(3) INPUT statement which inputs more than two variables ...Example 2

More than two variables can be written in an INPUT statement. (More than two INPUT statements can be arranged in one statement as shown below.)

```
10  INPUT X )
20  INPUT Y }  ——→ 10 INPUT X, Y, Z
30  INPUT Z )
```

Punctuate variables with commas.

When you execute this INPUT statement, " ? " is displayed at first to request the input of the value of X. After the value of X is entered, the values of Y and Z are requested in turn. After the value of Z is entered, this INPUT statement is terminated.

When input is performed with X = 123, Y = 456, and Z = 789, the display becomes as follows.

CLS RUN ⏎
1 2 3 ⏎
4 5 6 ⏎
7 8 9

```
RUN
?123
?456
?789
```

Variables, such as numerical variables, character variables, etc. can be used with all combinations and sequences in this kind of INPUT statement as shown below.

```
10 INPUT A$, X
```

However, a " , " (comma) must be used for punctuation between variables.

(4) INPUT statement that displays a message .... Examples 3 and 4

If a character string enclosed with " " (double quotation marks) is inserted between INPUT and a variable, the character string is displayed as it is. This is called a prompt statement (message). Incorrect input can be reduced by clarifying data for input with this message.
Execute the INPUT statement of example 3.

```
10  INPUT "AGE"; A
```

Then the following is displayed.

CLS SHIFT PO 0

```
AGE?_
```

Next, this INPUT statement is changed as follows.

```
10  INPUT "AGE", A
```

When you run this program, the display is as follows.

CLS RUN ↵

```
RUN
AGE_
```

When " ? " (input request display) should appear after a message, a " ; " (semicolon) is used for punctuation between the message and the variable.

This INPUT statement can be changed to write two variables or more.

```
10  INPUT "HEIGHT=" ;X, "WEIGHT=";Y
```

It can be considered that the following two INPUT statements were arranged into one INPUT statement.

```
10  INPUT "HEIGHT=";X }
                       } →10 INPUT "HEIGHT=";
20  INPUT "WEIGHT=";Y }     X, "WEIGHT=";Y
```

When you run this program, the display is as follows.

CLS SHIFT P0

175 ↵

65

```
HEIGHT=?175
WEIGHT=?65_
```

■ **Number of characters in character string used for a message**

The maximum number of characters for a character string enclosed with " " is 79 characters including the line numbers and INPUT command.

LET

Function	Assigns a data to a variable.
Format	LET variable = expression

The LET command placed at the beginning of an assignment statement is generally used in the following formats.

```
(Example 1)  LET A=10           LET A$="GAME"
(Example 2)  LET X=SIN (S-PI/4)     LET X$=A$+B$
```

The assignment statement assigns the value of the expression on the right side of the = sign to the variable on the left side of the = sign. A numerical expression corresponds to a numerical variable, and a character expression corresponds to a character variable. If the correspondence is not correct, a TM error is displayed.

■ **The tolerance range of the value assigned to a numerical variable.**

A numerical value that exceeds 10^{100} cannot be assigned to a numerical variable.

■ **The tolerance range of a number of characters assigned to a character variable.**

When the left side is a fixed character variable — up to 7 characters.
When the left side is a registered character variable — up to 16 characters.
When the left side is a fixed-length character array variable — up to 16 characters.
When the left side is a defined-length character array variable — up to 79 characters. (It depends on a definition.)

■ **LET can be omitted.**

`10  LET  A=1` is the same as `10  A=1`

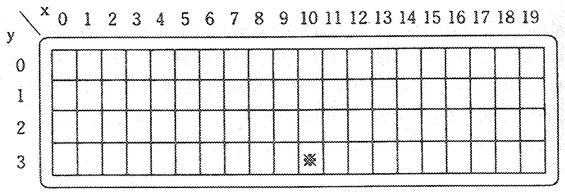
LOCATE

Function	Specifies the cursor position.
Format	LOCATE X, Y 0≦X<20, 0≦Y<4

Explanation

The display screen of the PB-700 is provided with 20 x 4 display positions as shown below. The display is generally performed from the left end of the screen by executing a PRINT statement. However, the display position can be freely changed by using the LOCATE command.

For example, when you specify LOCATE 10, 3, the cursor position are specified to (10, 3).



Usage

List A

```
10  X=1
20  X=X+1
40  PRINT "X=";X
50  GOTO 20
```

List B

```
10  X=1
20  X=X+1
30  LOCATE 0,0
40  PRINT "X=";X
50  GOTO 20
```

When you execute List A, the following is displayed.

```
X=2
X=3
X=4
X=5
```

⇧ Numerals appear continuously from the bottom of the screen and disappear toward the top of the screen.

When you add line 30 as shown in List B, the display is changed in such a way that only the "X = Numeral" is gradually increased at the top left corner of the screen.

PRINT/LPRINT

Functions	PRINT: Performs output to the display. LPRINT: Performs output to the printer.
Formats	PRINT expression [, expression] PRINT expression [; expression] LPRINT expression [, expression] LPRINT expression [; expression]

The PRINT and LPRINT commands are almost the same with the only difference being that output is either to the display or to the printer. However, when they are used with the TAB function, some differences occur.

Usage

Different kinds of data such as characters, numerical expressions, etc., as well as numerical values, can be displayed by using a PRINT statement.
As an example, it can be used as follows.

```
PRINT 1.414↵
PRINT A*B-2↵ .....
```

Since A and B are variables, computation result output is performed.

When these PRINT statements are executed, line change is performed after data is displayed. Run the following program as an example.

```
10  A=0:B=3
20  PRINT 1.414
30  PRINT A*B-2
40  INPUT C
```

Then the following is displayed.

CLS RUN ↵

```
RUN
 1.414
-2
?
 _ ....Cursor
```

A PRINT statement can display a plural number of expressions or character strings by using commas (,).

```
10  A=0:B=3
20  PRINT 1.414,A*B-2
30  INPUT C
```

When you run this program, line change is performed each time data is displayed, the same as the preceding display.

```
RUN
 1.414
-2
? _
  ˙····· Cursor
```

Although the display screen of the PB-700 consists of 4 lines, if output is performed on the 4th line, each line is rolled up.

```
10  PRINT 1, 2, 3, 4
20  END
```

```
 4
              RUN )
                1 |
Ready P0        2 } are rolled up.
                3 |
_               4 )
```

Although output is performed with line change by using a comma (,) for punctuation as mentioned above, if a plural number of expressions and character strings are punctuated with semicolons (;), continuous display is performed as follows.

```
10  A=0:B=0
20  PRINT 1.414;A*B-2
30  END
```

```
RUN
 1.414-2
Ready P0
_
```

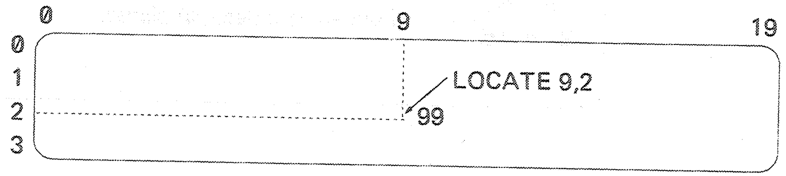
An easy-to-read message can be given by using this method as follows.

```
10  PRINT "ANSWER=" ; A*B-2
20  END
```

```
RUN
ANSWER=-2
Ready P0
_
```

Character and numerical output can both be performed with left justification as shown in the previous output examples. However, since the output of a numerical value is performed by including one position for a sign, if it is a positive value, a space occurs where the + sign is omitted. A LOCATE command and functions such as TAB and USING, with which the display location and format can be specified, are used for a PRINT statement.

```
10  CLS
20  A=1
30  LOCATE 9, 2
40  PRINT A
50  A=A+1
60  IF A<100 THEN 30
70  END
```



Numerical values 1~99 are continuously displayed at the location of the coordinates (9,2) of the character screen as displayed above. The space for the sign is displayed at the (9,2) coordinate point.

```
10  A=13:B=5:C=38
20  PRINT TAB(5);A;TAB(10);B;TAB(15);C
30  END
```

```
RUN     ┌Space for the sign┬──────────────────┐
         △13            △5                 △38
--TAB (5)-->
---------TAB (10)---------->
---------------------------TAB (15)------------->
0 1 2 3 4              10                15         19
```

Display is performed starting from the location specified by the TAB function when using a PRINT statement together with the TAB function.

Display can be performed with a uniform format in accordance with the USING function format as shown below.

```
10  A=3.1415:B=31.415:C=314.15
20  PRINT USING "###.##" ; A
30  PRINT USING "###.##" ; B
40  PRINT USING "###.##" ; C
50  FOR I=1 TO 1000:NEXT I
60  END
```

```
   3.14
  31.42
 314.15 ..........Rounded to 2 decimal places.
```

SAMPLE PROGRAM

* A program for displaying the radius, area, and circumference of a circle with a uniform format.

```
10 CLS
20 PRINT " R";TAB(7);"S";TAB(16);"L"
30 FOR R=1 TO 9
40 S=PI*R^2
50 L=2*PI*R
60 PRINT R;
70 PRINT "   ";:PRINT USING"###.##";S;
:PRINT "   ";:PRINT USING"##.##";L
80 NEXT R
90 END
```

SEE LOCATE, TAB, USING

PUT

Function	Stores variable data on a cassette tape.
Formats	PUT variable [, variable] PUT "file name" variable [, variable]

The PUT command stores variable data on a cassette tape. The format of the data file to be stored is ASCII. The GET command is used to read the data file.

(1) PUT A [, B, C . . . .]
(2) PUT "SALES" A [, B, C . . . .]

As shown in the above format (1), file name may be omitted. In this case, the file name should not be specified in the associated GET command. By separating two or more variables with a comma (,), it is possible to store plural variables as a data file by a single PUT command. Variables after PUT are stored on a 'first come, first served' basis. Numerical data is output in the same manner as it is output to the screen without using the USING function.

Usage

The following program stores the contents of array variables A(0) to A(10) by a PUT command. It is assumed that the array variables contain data.

```
10 REM PUT
20 DIM A(10)
30 FOR K=1 TO 10
40 A(K)=K
50 PUT A(K)
60 NEXT K
70 END
```

The following program reads the data stored on a cassette tape into the PB-700 using a GET command.

```
10 REM GET
20 DIM B(10)
30 FOR K=1 TO 10
40 GET B(K)
50 PRINT "B(";K;")=";B(K)
60 NEXT K
70 END
```

In this example, a FOR~NEXT loop is used to move the contents of array A to B. When storing plural data items, pay attention to the sequence in which they are stored.

SEE GET, SAVE

READ/DATA/RESTORE

Functions	READ: Reads data from a DATA statement into a variable. DATA: Stores data (constants, characters) in a program to be read by a READ statement. RESTORE: Changes the execution sequence of DATA statement.
Formats	READ variable [, variable] DATA data [, data] [, "character data"] RESTORE RESTORE line number (1 ≦ line number < 10000)

A READ statement is used with a DATA statement.
When a READ statement is executed, data is read from a DATA statement into a variable on a one to one basis.

Usage

The following is the simplest example of a READ/DATA statement program.

```
10 READ A
20 READ B$
30 PRINT A;B$
40 DATA 7,"B"
50 END
```

When you run this program, 7 is assigned to variable A, and "B" is assigned to character variable B$. The format of a variable and data must match.
Since as many variables as desired can be written continuously in a READ statement, line 10 and line 20 can be written on one line as follows.

```
10 READ A, B$
```

Character data may be enclosed with double quotation marks (" ") as mentioned above, or may not be enclosed with double quotation marks as follows.

```
40 DATA 7, B
```

However, if it is not enclosed with double quotation marks, a space is ignored as data. Therefore, if a space is required, it must be enclosed with double quotation marks.

```
DATA 7, "␣", "␣A", ␣C
                   └── This space is ignored.
```

Double quotation marks (") and commas (,) cannot be written in a character data except for the above format.

Although variables in a READ statement must correspond to DATA statements on a one to one basis, any number of variables or data can be placed in each statement.

```
10 CLEAR
20 DIM C(10)
30 READ A, B
40 FOR I=1 TO 10
50 READ C(I)
60 NEXT I
70 DATA 1,2,3
80 DATA 4,5,6,7,8,9
90 DATA 10,11,12
```

When you run this program, data is assigned to each variable as follows.

```
A  B  C(1)  C(2)  C(3)·········C(8)  C(9)  C(10)
↓  ↓   ↓     ↓     ↓            ↓     ↓     ↓
1  2   3     4     5············10    11    12
```

If the number of data is less than the number of variables to which data are assigned by a READ statement, an error (DA error) occurs. However, if the number of data is more than the number of variables, an error does not occur but the extra data is ignored.
A DATA statement can be placed before a READ statement.
A DATA statement in which data is read by a READ statement can be specified by using a RESTORE statement.

A RESTORE statement has two different formats, one in which the line number is written, and another in which the line number is not written. If the line number is not written, when RESTORE is executed, the following READ statement reads data from the first DATA statement.

```
10  READ A, B
20  RESTORE
30  READ C, D
40  PRINT A;B;C;D
50  DATA 7, 2
60  END
```

When you run this program, the assignments performed are A = 7, B = 2, C = 7 and D = 2.
The DATA statements can also be specified by specifying the line number.

```
10  RESTORE 50
20  READ A, B
30  PRINT A;B
40  DATA 3.7, 6.5
50  DATA 7.1, 9.3 ········ DATA statement specified by RESTORE
                           statement on line 10.
60  DATA 5, 10.2
70  END
```

When you execute this program, A = 7.1 and B = 9.3 are performed as the assignment.
Variables and numerical expressions can be used for a line number specification of a RESTORE statement.

Precautions should be taken when a READ statement is used in a program which has moves to a plural number of program areas.

```
P0  10  READ A,B            P1  10  READ X, Y
    20  GOTO  PROG 1            20  PRINT X;Y
    30  DATA 1,2,3,4            30  DATA 71,65
                                40  END
```

When you execute this program, displayed data are not 71 and 65, but are 3 and 4. In other words, although the execution of this program has been shifted to P1 by GOTO PROG 1, the DATA statement of P0 is still used.
This is useful when the DATA statement of a main program is used in a subroutine.
If it is necessary to read 71 and 65 into X and Y in this program, specify the line number at the beginning of Program P1 as follows.

```
5  RESTORE 30
```

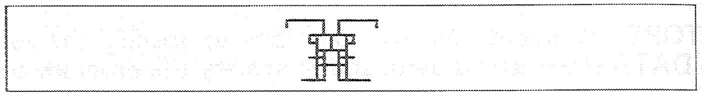
A RESTORE statement has two functions to specify the sequence in which a DATA statement is used, and to specify the program area.

SAMPLE PROGRAM

* Alien movement program.

```
  10 CLS
  20 N=0
  90 FOR I=1 TO 58
 100 READ X,Y
 110 IF X=0 THEN RESTORE :N=N+17
 120 IF N>=140 THEN END
 130 DRAW(X+N,Y)
 140 NEXT I
 143 CLS
 145 GOTO 90
 150 DATA 5,15,6,15,7,15,8,15,9,15,10,1
5
 160 DATA 12,15,13,15,14,15,15,15,16,15
,17,15
 170 DATA 5,16,10,16,12,16,17,16
 180 DATA 8,17,9,17,10,17,11,17,12,17,1
3,17,14,17
 190 DATA 8,18,9,18,11,18,13,18,14,18
 200 DATA 9,19,10,19,11,19,12,19,13,19
```

```
  210 DATA 8,20,9,20,10,20,12,20,13,20,1
4,20
  220 DATA 9,21,10,21,11,21,12,21,13,21
  230 DATA 9,22,10,22,12,22,13,22
  240 DATA 7,23,8,23,9,23,10,23,12,23,13
,23,14,23,15,23,16,23
  250 DATA 0,0
```



In this program, a character moves from the left to the right on the screen.
DATA statements from line 150 to line 250 are the dot coordinates to draw one character.
One character is drawn, then it is erased on line 143, data is restored on line 110, and the location is changed to draw the same character.
When it moves to the right side of the screen by repeating this procedure, the program is terminated.

REM

Function	Gives a comment to a program.
Format	REM comment statement

Unlike other commands, the REM command does not execute anything. Since anything can be freely written after REM, a program explanation can be written at important points in a program as shown below so that the content of each part of a program can be understood by looking at a list.

```
 10  REM  *PROCESSING RESULTS*
 20  DIM  A(100)
          :
          :
100  REM  *PRINT OUT*
          :
          :
500  END
```

Since all of the characters and symbols written after REM are considered to be comments, other commands cannot be continued by a multistatement.

STOP

Function	Stops program execution.
Format	STOP

If the STOP command is found in a program during program execution, a STOP message is displayed which suspends program execution. The execution of a program stopped by the STOP command can be started from the instruction next to STOP by inputting the CONT command.

Usage

Let's check the function fo the STOP command in the following program.

```
10 A=1:B=5
20 C=A+B
30 STOP
40 PRINT C
50 END
```

When you execute this program, the following is displayed.

```
STOP P0 – 30
```

This indicates that execution is suspended by the STOP command on line 30 of program area P0.
In this state, the content of the variable can be checked as follows.

A ENTER → 1 Displays the value of A.
C ENTER → 5 Displays the value of C.

Also, an optional value can be assigned to the variable by entering

C = 0 ↵

It is actually used to compulsorily stop a program by inserting a STOP command at the part where the operation is doubtful during debug to confirm the content of a variable.

1 Execution can be resumed by using CONT even if the following operation are performed during STOP command execution.
(1) Manual calculation.
(2) Assignment to a variable (Assignment without using LEN).
(3) Confirmation of the content of a variable.
(4) Execution of the following commands.

ANGLE, BEEP, CLEAR, CLS, DIM, ERASE, PRINT, LPRINT, TRON, TROFF

2 If the following operations are executed during STOP command execution, execution resumption by CONT cannot be performed.
(1) Execution of manual commands (EDIT, SAVE, LOAD, LIST, etc.)
(2) Execution of PUT/GET.
(3) When an error occurs.

SEE CONT

TRON/TROFF

Function	Traces program execution and terminates tracing of program execution.
Formats	TRON TROFF

Explanation

TRON and TROFF commands are used during program debug.

TRON .... Specifies the trace mode.
TROFF ...... Releases the trace mode.

When the trace mode is specified, a program is executed while the present program area number, and the line number are displayed as follows.

```
Display 〔0:110〕
          |  └──── Line number being executed
        Program area being executed
```

Since TRON, TROFF are program commands, they can be used by writing them in a program, however they are usually used by direct entry.

Usage

Input the following program and execute it.

```
P0
   10  BEEP 1
   20  GOTO PROG 1
P1
   10  BEEP 0
   20  GOTO PROG 0
```

When you run this program, two beeps start sounding alternately. Now, enter TRON ⏎ to specify the trace mode after pressing the BRK key. After entry completion, run the program. The program area and line number now being executed are continuously displayed with the display as shown below.

```
〔0 : 10〕        〔0 : 20〕
〔1 : 10〕        〔1 : 20〕
〔0 : 10〕        〔0 : 20〕
```

Now, you will find that the execution speed is slow in the trace mode since the spacing between BEEP sounds is fairly large.
In addition, during INPUT statement execution, it stops by displaying "?" after the [area number, line number]. And, the result can be displayed during PRINT statement execution. This command is very convenient during debug because the program execution process can be traced as mentioned above.

4-3 NUMERICAL FUNCTIONS

SIN

Function	Gives the sine of X, Sin X.
Format	SIN numerical expression $-5400° < \text{Numerical expression} < 5400°$

The SIN function computes Sin X.
X can be used by selecting one of 3 angle units (DEG, RAD, GRA).
When the power is turned on, the angle unit is set to DEG (degree).

Example

SIN X computation is performed by using DEG (degree).

```
10 PRINT SIN30
20 PRINT SIN45
30 PRINT SIN90
50 END
```

When you run this program (R U N ↵ or SHIFT RUN O ↵), the results for SIN 30, SIN 45 and SIN90 are displayed as if they are flowing and disappear.
Run this program again by adding the following line.

```
25 STOP
```

Now the results for SIN30 and SIN45 are displayed and stopped.

Execution Example

```
 0.5
 0.7071067812
STOP P0-25
 _
```

To observe it continuously, press either CONT ↵ or SHIFT CONT W ↵, then the next result (SIN90) is displayed. (See page 136.)

Execution Example

```
 1
Ready P0
_
```

Next select one of the angle units (DEG, RAD, GRA) and compute SIN X.

```
10 REM SIN X  EXAMPLE
20 PRINT "ANGLE=";
30 INPUT K
40 ANGLE K
50 PRINT "SIN X  X=";
60 INPUT X
70 PRINT "SIN";X;"=";SINX
80 STOP
90 END
```

When you run this program, the angle unit is requested first as follows.

ANGLE = ?

Next you should specify the angle unit as follows.

ANGLE 0 → DEGREE
ANGLE 1 → RADIAN
ANGLE 2 → GRADIENT

RADIAN is selected as an example.
Make an entry of " 1 ".

SIN X   X = ?   is now requested.

Next, when you input the Radian angle, such as PI/4, the display result is as follows.

Execution Example

```
SIN 0.7853981634=0.
7071067812
STOP P0-80
_
```

Although the angle unit is modified by using the ANGLE command as described, the input range for each angle unit is restricted as follows.

DEG $-5400^\circ < \text{Numerical expression} < 5400^\circ$
RAD $-30\pi < \text{Numerical expression} < 30\pi$
GRA $-6000 < \text{Numerical expression} < 6000$

When the value of a numerical expression is outside the range shown above, an error (BS error) occurs.
A variable and a numerical expression, as well as a real number (such as 30), can be used for the argument.
When only one real number or variable is used, the argument may be placed or not placed inside parentheses. However, when a numerical expression is used, the result will differ depending on whether it is placed or not placed inside parentheses as follows.
Therefore, precautions shall be taken.

SIN X + Y ..... Add Y to the result of the SIN X computation.
SIN (X + Y) .... Compute the SIN of the result of X + Y.

SEE ANGLE, COS, TAN

Memorandum

There are three different ways to express the angle of a trigonometric function which are "Degree (DEG)", "Radian (RAD)" and "Gradient (GRA)."

DEG .... 1° is 1/360 of the circumference of a circle.
RAD .... 1 rad. is $1/2\pi$ of the circumference of a circle.
GRA .... 1 gra. is 1/400 of the circumference of a circle.

Degree and Radian are mainly used with the following relationship.

$$1^\circ = \pi/180\text{rad} = 3.141592654/180 \text{ rad}$$

COS

Function	Gives the cosine of X, Cos X.
Format	COS numerical expression $-5400^\circ < \text{Numerical expression} < 5400^\circ$

The COS function is used to compute COS X.
The angle units for COS X, the X argument input range, and the precision are exactly the same as for SIN X.

Example

Provide input to a program in which COS X is used.

```
10 REM COS X EXAMPLE
20 PRINT "ANGLE=";
30 INPUT K
40 ANGLE K
50 PRINT "COS X  X=";
60 INPUT X
70 PRINT "COS";X;"=";COSX
80 STOP
90 END
```

When you run this program, angle unit input is requested by;

ANGLE = ?

Next, if the gradient angle unit is to be used, input "2". After this, since the angle is requested as follows, provide an input of 1355.

COS X X = ?

The result is −0.7604059656.

SEE SIN, ANGLE, TAN

TAN

Function	Gives the tangent of X, Tan X
Format	TAN numerical expression $-5400° <$ Numerical expression $< 5400°$

The TAN function is used to compute TAN X.
The angle units are the same as for SIN X and COS X.

Example

Provide a program to obtain TAN X.

```
10 REM TAN X EXAMPLE
20 PRINT "ANGLE=";
30 INPUT K
40 ANGLE K
50 PRINT "TAN X  X=";
60 INPUT X
70 PRINT "TAN";X;"=";TANX
80 STOP
90 END
```

When you obtain TAN 45 by using the DEG angle unit (ANGLE 0), the result is displayed as;

TAN 45 = 1.

Next, if you try to obtain TAN 90, an "MA error" is displayed.
When the TAN function is used, the value of TAN X suddenly increases as it approaches 90°.
If it is TAN 90, the value becomes infinite and computation cannot be performed.
As a result, an "MA error" was displayed in the above example. When the value of X is $\pm 90 * (2n\text{-}1)$ (n is an integer) in TAN X, precautions shall be taken since an error occurs as mentioned above.

SEE SIN, COS, ANGLE

ASN, ACS, ATN

Functions	ASN gives the arcsine, Sin^{-1} X. ACS gives the arccosine, Cos^{-1} X. ATN gives arctangent, Tan^{-1} X.
Formats	ASN X, ACS X, ATN X. $\lvert X\rvert \leqq 1$ (ASNX, ACS X) $\lvert X\rvert \leqq 10^{100}$ (ATN X)

The ASN, ACS, and ATN functions are used to compute inverse trigonometric functions such as SIN^{-1} X, COS^{-1} X and TAN^{-1} X.
The trigonometric functions (SIN, COS, TAN) are used to give angles and obtain their trigonometric function values. On the other hand, the inverse trigonometric functions obtain the angles when the trigonometric function values are given.

Example

A program example which uses ASN X is shown below.

```
10 REM ASN X EXAMPLE
20 PRINT "ANGLE=";
30 INPUT K
40 ANGLE K
50 PRINT "ASN X  X=";
60 INPUT X
70 PRINT "ASN";X;"=";ASNX
80 STOP
90 END
```

When you run this program, the following two input requests are displayed.

ANGLE = ? 0 ↵ ...... Specifys "degree."
ASN X X = ? 1 ↵ ...... Inputs the trigonometric function value.

When you input these values, the following is displayed.

ASN 1 = 90

In other words, the angle X of SIN X = 1 was obtained.

Try this program again using ACS and ATN to replace ASN.
These inverse trigonometric functions are specified by ANGLE the same as for SIN, COS, and TAN.
The degree (DEG) angle range in the computation result is as follows.

$$-90^\circ \leqq \mathrm{ASN} \leqq 90^\circ$$
$$0^\circ \leqq \mathrm{ACS} \leqq 180^\circ$$
$$-90^\circ \leqq \mathrm{ATN} \leqq 90^\circ$$

Since the TAN X value is 1 or more, the argument X of ATN X is a value in almost all ranges.
However, when

$$X \geqq 2E10,$$

ATN X = 90°
Since SIN X and COS X do not theoretically exceed 1, the value of argument X of ASN X and ACS X must not exceed 1.

SEE SIN, COS, TAN, ANGLE

SQR

Function	Gives the square root of the argument.
Format	SQR numerical expression Numerical expression ≧ 0

The SQR function is used to obtain a square root as follows.

SQR X = X^0.5 = $\sqrt{X}$

In this case, the value of X must be larger than 0.

Example

The following program is used to input the area of a circle in order to obtain the circular radius.

```
10 REM SQR X EXAMPLE
20 PRINT "CIRCLE AREA=";
30 INPUT S
40 R=SQR(S/PI)
50 PRINT "CIRCLE RADIUS=";R
60 LOCATE 0,2
70 END
```

When you execute this program, the following is requested.

CIRCLE AREA = ?

Enter 100 as an example, then 5.641895835 is displayed which is the value for the circle radius.
If you run this program again by entering a minus value, an MA error will be immediately displayed because when the argument X of SQR X is a minus value, SQR (S/Pi) becomes an imaginary number. To avoid this error, it is recommended that the following line be added to check for a positive or negative argument.

```
35 IF S<0 THEN 20
```

LOG, LGT

Functions	LOG X . . . . Gives the value of natural logarithm, $\log_e X$. LGT X . . . . Gives the value of common logarithm, $\log_{10} X$
Formats	LOG numerical expression LGT numerical expression Numerical expression > 0

LOG X computes the value of natural logarithm, $\log_e X$ (lnX). In this case, "e" is the base of a natural logarithm.
The value of e is as follows.

e = 2.718281828 . . . . . .

LGT X computes the value of common logarithm, $\log_{10} X$. The base of a common logarithm is 10.

Example

The following program computes LOG X by gradually giving the value of X.

```
10 REM LOG X EXAMPLE
20 PRINT "X=";
30 INPUT X
40 PRINT "LOG";X;"=";LOGX
50 GOTO 20
```

When you execute this program,

X = ?

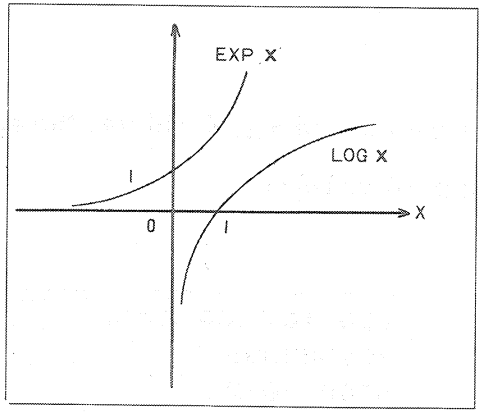
is displayed which requests the value of X for LOG X.

If you enter "1", the value of LOG X is displayed as LOG (1) = 0, and next value of X is requested.

Execution Example

CLS	RUN
RUN ↵	X=?
1 ↵	LOG 1=0
	X=?_

The logarithmic function LOG X has an inverse relationship with the exponential function EXP X as shown in the following graph.



X > 0 is required in a logarithmic function as shown in the above graph.
If a negative value is entered, an MA error is displayed.
While LOG X is the logarithm of X, which has a base of e, the logarithm $\log_y X$, which has a base other than e, can be computed by the following formula.

LOG X/LOG Y

Therefore the common logarithm of X, LGT X can also be obtained with the following formula.

LOG X/LOG 10

SAMPLE PROGRAM

* This program allows many different logarithmic values to be obtained when base values are entered.

```
    10 REM ***LOG X/LOG Y***
    20 PRINT "X=";
    30 INPUT X
    40 PRINT "Y=";
    50 INPUT Y
    60 PRINT "LOG";X;"/LOG";Y;"=";LOGX/LO
GY
    70 LOCATE 0,3
    75 STOP
    80 END
```

This program computes the value of $\log_Y X$ with the formula, LOG X/LOG Y.
An execution example is provided below.

Execution Example

```
RUN ↵     LOG 10/LOG 2=3.
 10 ↵     321928095
  2 ↵     STOP P0-75
          _
```

SEE EXP

EXP

Function	Gives the exponentional function e^x.
Format	EXP numerical expression Numerical expression ≦ 230

The EXP function is used to compute the value (e^x) of an exponential function. The value of "e", which is the base of an exponential function, is as follows.

$e = 2.718281828\ .....$

Since an "exponential increase" is often mentioned, the nature of this function is that as the value of argument X increases, the value of EXP X suddenly increases.

Example

Enter the following program to observe the change in the value of EXP X.

```
    10 REM EXP X EXAMPLE
    20 PRINT "A=";
    30 INPUT A
    40 FOR X=1 TO A
    50 PRINT "EXP";X;"=";EXPX
    60 FOR I=1 TO 300:NEXT I
    70 NEXT X
    80 END
```

When you execute this program, a request is made for the maximum argument value X of EXP X.

A = ?

Enter "10" as an example and press the ↵ key, and the results shown on the following page are continuously displayed.

Execution Example

```
EXP 1= 2.718281828
EXP 2= 7.389056099
EXP 3= 20.08553692
EXP 4= 54.59815003
EXP 5= 148.4131591
EXP 6= 403.4287935
EXP 7= 1096.633158
EXP 8= 2980.957987
EXP 9= 8103.083928
EXP 10= 22026.46579
```

You will see that the value of EXP X increases suddenly.
Run this program again by entering "231" for A.
The value of EXP X is continuously displayed, and when the value of the argument is 231, an MA error occurs.
The input range of the argument X of EXP X is actually

$$X \leqq 230.2585.$$

The value of EXP X with X = 230.2585 is as follows.

Execution Example

```
EXP 230.2585= 9.999907006E99
```

An overflow almost occurs based on the result mentioned above as you can see.

ABS

Function	Gives the absolute value of the argument.
Format	ABS numerical expression

ABS X gives the absolute value of X which is mathematically expressed as follows.

ABS X = |X|

In regard to the X of ABS X,
When $X \geqq 0$ (value X is positive) . . . . ABS X = X,
and when $X < 0$ (value X is negative) . . . . ABS X = –X.
In other words, ABS X computes in a way that allows the result to become a positive number (absolute number).

Example

Now let's look at a program which uses the ABS function.

```
   10 REM ABS X EXAMPLE
   20 READ A,B,C,D
   30 X=A:GOSUB 40:X=B:GOSUB 40:X=C:GOSU
B 40:X=D:GOSUB 40:END
   40 PRINT "ABS";X;"=";ABSX
   50 FOR I=1 TO 200:NEXT I
   60 RETURN
   70 DATA5,-5,0,-7.5
```

This program reads 5, –5, 0 and –7.5 into variables A, B, C and D respectively by using a READ statement and computes ABS A to ABS D.

The result is displayed as follows.

Execution Example

```
ABS 5=5
ABS-5=5
ABS 0=0
ABS-7.5=7.5
```

An operation which is the same as the ABS function can be performed by using the SGN function as follows.

ABS X ← equal → X ∗SGN X

SAMPLE PROGRAM

∗ A program in which no error occurs when a negative value is entered.

```
10 INPUT X
20 S=SQR(ABSX)
30 L=LOG(ABSX)
40 PRINT "SQRX=";S
50 FOR I=1 TO 200:NEXT I
60 PRINT "LOGX=";L
70 END
```

When an argument X is a negative value for functions such as SQR X and LOG X, an MA error occurs.
Therefore, in this program, computing is performed by using the absolute value of X.

SEE SGN

INT

Function	Gives the largest integer which does not exceed the argument value.
Format	INT numerical expression

INT X gives the largest integer that does not exceed the value of X. For example, when the values of X are 3.9, 0.5, –0.5 and –3.9, INT X is as follows for each of these values.

INT 3.9 = 3
INT 0.5 = 0
INT –0.5 = –1
INT –3.9 = –4

When the value of the argument is positive, the value after the decimal point is discarded. However, when it is negative, precautions should be taken. For example, if the value is –0.5, it is not 0 but –1 which is the largest integer that does not exceed –0.5.

Example

Let's try the following program.

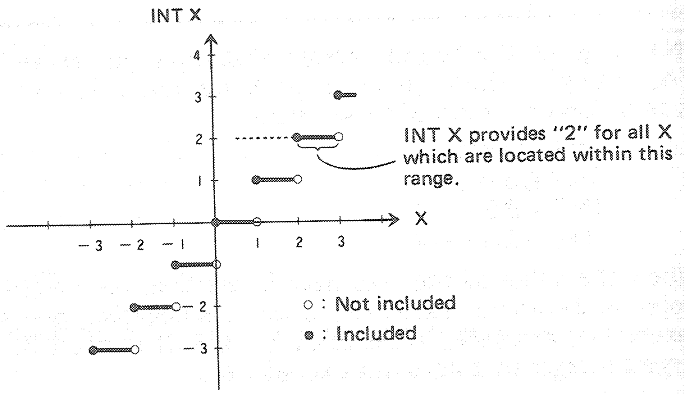
```
    10 REM INTX EXAMPLE
    20 READ A,B,C
    30 X=A:GOSUB 40:X=B:GOSUB 40:X=C:GOSU
B 40:END
    40 PRINT "INT";X;"=";INTX
    50 FOR I=1 TO 200:NEXT I
    60 RETURN
    70 DATA5.3,0.5,-3.9
```

Execution Example

```
INT 5.3= 5
INT 0.5= 0
INT-3.9=-4
```

When you run this program, the result shown on the previous page is displayed.
An INT function graph is drawn as follows by placing values of X horizontally and values of INT X vertically.

INT X Graph



INT X provides "2" for all X which are located within this range.

The difference for positive or negative values can be found by using this graph.
The INT function is often used by combining it with other functions such as the RND function. In addition, the FRAC function, ROUND function, etc. are similar to the INT function.

SAMPLE PROGRAM

* A program which displays 5 integers from 0 to 9 at random.

```
10 FOR I=1 TO 5
20 PRINT INT(10*RND);
30 NEXT I
40 END
```

This program was prepared by combining the INT function with the RND function as an example.

SEE FRAC, ROUND, RND

FRAC

Function	Gives the value of the fractional part of the argument.
Format	FRAC numerical expression

FRAC X gives the value of the fractional part of X.
Simple examples are provided as follows.

FRAC 1.123 = 0.123
FRAC −1.123 = −0.123

It simply functions to discard the integer part as shown above.

Example

Try the following program by entering many different values.

```
10 REM FRAC X  EXAMPLE
20 PRINT "NUMBER";
30 INPUT X
40 PRINT FRAC X
50 GOTO 10
```

SAMPLE PROGRAM

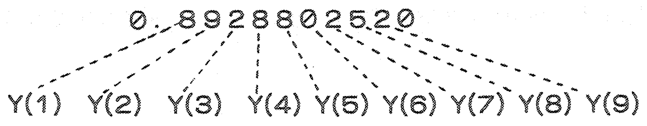
* A program which fetches a 9 digit random number after the decimal point that is generated by the RND function as an integer with one digit.

```
10 DIM Y(9)
20 X=RND
30 PRINT "X=";X
40 FOR I=1 TO 9
50 Y(I)=INT(10*X)
60 PRINT "Y(";I;")=";Y(I)
70 FOR J=1 TO 100:NEXT J
80 X=FRAC(10*X)
90 NEXT I
```

This program assigns a value generated by the RND function to variable X.

The value is multiplied ten times to obtain a one digit integer with the INT function which is assigned to the array variable Y(1) in which the residual value after the decimal point is fetched with the FRAC function and assigned as a new replacement value for X. This procedure is repeated 9 times to distribute a one digit random number to array variables Y(1) to Y(9).

For example, if the random numbers shown below are generated, they are distributed as follows.



SEE INT, RND, ROUND

SGN

Function	Gives a plus or minus sign according to the argument.
Format	SGN numerical expression

The SGN X function judges whether the value of argument X is positive or negative. SGN X provides three different results as follows.

X > 0 (If positive) .... SGN X = 1
X = 0 (If 0) .... SGN X = 0
X < 0 (If negative) .... SGN X = −1

Example

Try the following program.

```
10 REM SGN X EXAMPLE
20 PRINT "JUDGEMENT OF + OR -"
30 PRINT "NUMBER";
40 INPUT X
50 A=SGNX
60 IF A=1 THEN PRINT "+":GOTO 40
70 IF A=0 THEN PRINT "0":GOTO 40
80 IF A=-1 THEN PRINT "-":GOTO 40
```

On lines 60 to 80, if it is positive, a "+" is displayed, if it is negative, a "−" is displayed, and if it is 0, a "0" is displayed based on the value obtained by SGN X.

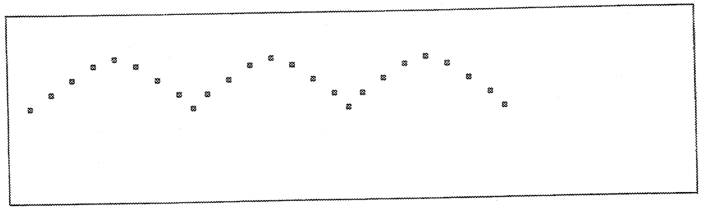
SAMPLE PROGRAM

* A program which provides a sine curve.

```
10 CLS :FOR X=0 TO 540 STEP 20
20 S=SGN(SINX)
30 Y=S*INT(S*10*SINX)
40 IF S<0 THEN 70
50 DRAW(X/4,16-Y)
60 GOTO 80
70 DRAW(X/4,16+Y)
80 NEXT X
90 END
```

This program obtaines the SIN X from 0 to 540 degrees with a 20 degree increase which is multiplied 10 times, and assigns it to variable Y as a 1 digit value.
Since the SGN function is used for the value of SIN X is derived, even if SIN (X) is negative, the integer portion can only be fetched by the INT function.
The values of X and Y that were obtained as mentioned above are plotted for the (X/4, 16–Y) coordinates when it is positive, and for the (X/4, 16+Y) coordinates when it is negative with dot coordinates (0, 16) to (159, 16) as its center by using the DRAW command.
See the Section DRAW.
When you run this program, a rough sine curve is plotted as shown below.

Execution Result



ROUND

Function	Gives the value of numerical expression 1 which is rounded at a position specified by numerical expression 2.
Format	ROUND (numerical expression 1, numerical expressin 2) Numerical expression 2: Digit position

ROUND (X, Y) gives the value of X which is rounded at the 10^Y position as follows to provide an example.

ROUND (12345, 2) = 12000

The numerical value 12345 is rounded at the 10^2 position which is a position of 100.

Example

You can confirm the operation of the ROUND function with the following program.

```
10 FOR Y=3 TO -5 STEP -1
20 X=12345.67891
30 Z=ROUND(X,Y)
40 PRINT USING"#####.#####";X,Z:PRINT
50 FOR J=0 TO 150:NEXT J
60 NEXT Y
70 END
```

This program performs a calculation in which the value Y of ROUND (X, Y) is decreased by 1 from 3 to –5.
The value specified for the 2nd argument Y is determined to be

$|Y| < 100$

and if it exceeds this, a BS error occurs.
When a value with a fraction is specified for the value of Y, the fractional part is discarded.

Execution Example

```
10000
12000
12300
12350
12346
12345.7
12345.68
12345.679
12345.6789
```

SAMPLE PROGRAM

* This program displays binary 8-bit random numbers and their decimal values.

```
 10 Y=0
 20 FOR I=7 TO 0 STEP -1
 30 X=ROUND(RND,-1)
 40 Y=Y+(2^I)*X
 50 PRINT X;
 60 NEXT I
 70 PRINT "=";Y
 80 LOCATE 0,1
 90 FOR J=0 TO 50:NEXT J
100 END
```

Since random numbers generated by the RND function (page 246) are rounded to one decimal place on line 30, the value of X is 0 or 1.

The value of X is generated 8 times to provide an 8-bit random number that consists of 0s and 1s.
At the same time, the binary 8-bit value is converted to a decimal number on line 40 which is continuously displayed. An execution example is provided below.

RUN ⏎

```
RUN
00101001 =41
```

PI

Function	Gives the ratio of the circumference of a circle to its diameter, π.
Format	PI

PI gives round numbers for the ratio of the circumference of a circle to its diameter, π.
The value is provided for π as follows

$\pi = 3.141592654\ldots\ldots$

Example

The following program computes the area of a circle.

```
10 REM PI EXAMPLE
20 PRINT "CIRCLE AREA"
30 PRINT "RADIUS";
40 INPUT R
50 S=PI*R^2
60 PRINT "S=";S
70 END
```

If you enter 5 as the radius value, the area of the circle is displayed as follows.

Execution example

```
S= 78.53981634
```

RND

Function	Gives a random number value.
Format	RND 0 < Random number < 1

The RND function gives a 10 digit random number value that is larger than 0 and smaller than 1.

A random number is one which occurs without any rule before and after generation, or one with a value that cannot be predicted.

Random numbers were first required for the simulation of statistical phenomena or probability models, and are now used for simulations such as economic forecasts or TV games. In particular, the fun provided by TV games is largely due to this random number function.

Usage

The following program generates and displays 10 random numbers.

```
10 FOR N=1 TO 10
20 PRINT RND
30 FOR X=1 TO 500:NEXT X
40 NEXT N
50 END
```

Although the result is as follows, when you run this program, it is natural that different random numbers will be generated.

```
0.6791506196
0.959823211
0.205719988
0.503905755
0.306977109
0.106577855
0.4177075471
0.501741468
0.755155195
0.456091832
```

Since a large number of digits are generated, they are not easy to handle as they are. Therefore, when it is used for a game, etc., it is used after providing random number values within an appropriate range by combining it with the INT function and ROUND function as follows.

(1) Fetch integers up to a desired digit.

INT (RND ∗ 10^L) ..... L indicates the number of digits.

(2) Fetch integers from N to the upper limit M.

ROUND (RND ∗ (M–N), –1) + N .... N and M are integers. (N<M)

4-4 CHARACTER FUNCTIONS

ASC

Function	Gives the decimal code for the first character of a character string.
Formats	ASC "Character string" ASC (Character variable)

All characters, numerals, and symbols displayed by this personal computer have a number which is called ASCII code.
Examples are as follows.

"A" 65
"B" .... 66 (See page 321, CHARACTER CODE TABLE.)
"6"..... 54

These character numbers (codes) can be directly determined by the personal computer when the ASC function is used, and can also be determined by using the CHARACTER CODE TABLE.

Example

Enter PRINT ASC ("E") ↵
The "E" character ASCII code
69
is displayed.

When an entry is made to determine two character codes or more such as

PRINT ASC ("EF") ↵

only the character code for "E", the initial character, is displayed. Therefore, to determine the codes for a long character string (such as "ABCDEF .... ") serially from the beginning, they can be displayed by a program that uses the MID$ function (see MID$).

SAMPLE PROGRAM

* A program that displays character codes continuously.

```
10 REM ASCII CODE
20 CLS
30 INPUT "WHICH CHARACTER";A$
40 PRINT ASC(A$)
50 GOTO 30
```

When character input was performed on line 30, the codes are displayed on line 40. Since it returns to line 30 on line 50, character input can be continuously performed to determine codes.

The execution result is as follows.

Execution Example

```
WHICH CHARACTER? A
 65
WHICH CHARACTER? T
 84
WHICH CHARACTER? R
 82
WHICH CHARACTER? 7
 55
WHICH CHARACTER? 1
 49
```

Since this program indefinitely requests character codes, press the BRK key to stop it.

SEE CHR$

CHR$

Function	Gives the character represented by a specified ASCII code.
Format	CHR$ (Code) 0 ≦ Code < 256

CHR$ is a function that determines a character (character, number, or symbol) by specifying ASCII code.

Example

Enter PRINT CHR$ (66) ⏎

Then the character "B" is displayed for ASCII code 66.
To determine two characters at one time, enter

PRINT CHR$ (71); CHR$ (80) ⏎

Then the characters "G" and "P" which correspond to ASCII codes 71 and 80 are displayed.
Characters that can be entered by direct PB-700 key input are numerals, capital alphabetic characters, small alphabetic characters, and some symbols. Other characters (such as symbols, graphics, Katakana characters, etc.) are displayed by using CHR$ (See the CHARACTER CODE TABLE). Numbers (codes) that can be specified by CHR$ are within a 0 ≦ Code < 256 range and the fractional part is ignored.

SAMPLE PROGRAM

```
  5 V=0
 10 FOR I=33 TO 254
 20 PRINT I
 30 V=V+1
 40 FOR J=1 TO 14
 50 PRINT CHR$(I);
 60 NEXT J
 70 PRINT
 80 IF V<3 THEN 110
 90 K$=INKEY$:IF K$="" THEN 90
100 V=0
110 NEXT I
120 END
```

* This program displays the characters for character codes 33 to 254. Each time this program is executed, character codes 33 – 35 are displayed by 14 characters, and it momentarily halts. Perform the entry of a key, then 14 characters of the following 3 code portion are displayed each time. When this operation is repeated and the display of the 254th character is terminated, the program is ended.
Some execution examples are shown below.

```
 33
!!!!!!!!!!!!!!
 34
""""""""""""""
 35
##############
 36
$$$$$$$$$$$$$$
 37
%%%%%%%%%%%%%%
 38
&&&&&&&&&&&&&&
```

SEE ASC

VAL

Function	Converts a character string into a numerical value.
Formats	VAL "Character string" VAL (Character variable)

VAL is a function that converts a character into a numerical value. While this function differs from other functions, the difference in a character and a numerical value must be explained in order to understand this function.
Compare the following two program examples.

Program (1)

```
10 READ A,B
20 C=A+B
30 PRINT C
40 END
50 DATA3,5
```

Program (2)

```
10 READ A$,B$
20 C$=A$+B$
30 PRINT C$
40 END
50 DATA3,5
```

In program (1), the numerical data is read into the numerical variables A and B, and the arithmetic result is displayed by assigning it to C.
The result of program execution is

␣ 8

In the above, ␣ indicates a space where an omitted "+" sign is to be inserted.
On the other hand, in Program (2), 3 and 5 are read into the character variables A$ and B$, respectively, as character data instead of numerical data.
In regard to character variable operations, only addition can be performed with the result assigned to C$.
When this program is executed,

35

is displayed. The result is just the display of a character string based on the fact that there is no "—" sign or a blank (one character area) where a "+" sign can be inserted.

This blank is very significant, and the difference will be clarified by comparing the Program (3) and (4) execution examples which follow.

Program (3)

```
10 FOR I=1 TO 10
20 READ A
30 PRINT A;
40 NEXT I
50 END
60 DATA3,8,-6,7,21
70 DATA223,18,8,1,0
```

Execution Example

```
3 8-6 7 21 223 18 8
1 0
```

Program (4)

```
10 FOR I=1 TO 10
20 READ A$
30 PRINT A$;
40 NEXT I
50 END
60 DATA3,8,-6,7,21
70 DATA223,18,8,1,0
```

Execution Example

```
38-67212231881 0
```

Usage

When an operation such as that in Program (1) is performed by using a numeral read into the character variables A$ and B$, as shown in Program (2), this program can be converted by using the VAL function for the following modification.

```
10 READ A$,B$
20 C=VAL(A$)+VAL(B$)
30 PRINT C
40 END
50 DATA3,5
```

When the program is executed, "␣8" is displayed as for Program (1). Precautions shall be taken as follows when the VAL function is used.

(1) When characters other than a numerical value, decimal point, sign (+, −), and exponent sign "E" appear in the character string, all the following characters are ignored. (When the exponent sign "E" appears twice or more, they are ignored.)
(2) The first space in a character string is skipped.
(3) When the initial character of a character string is not a numerical value, decimal point, or a sign, or when a character string is only a sign and a decimal point, 0 (zero) is provided.
(4) When more than three numerals exist after an exponent sign "E", an SN error occurs.

SAMPLE PROGRAM

* An input subroutine where no error occurs when any key is pressed as a response to the input request of Menu Nos. 1—5.

```
100 Z$=INKEY$:IF Z$="" THEN 100
110 IF Z$<"1" THEN 100
120 IF Z$>"5" THEN 100
130 GOSUB VAL(Z$)*1000
140          ≀
```

In this program, when a key from 1 to 5 is pressed, the program jumps to each subroutine of lines 1000 to 5000, and when a key other than one of these is pressed, re-input is requested in which no error occurs. It is convenient to use as an input routine for Menu selection.

SEE STR$

STR$

Function	Converts a numerical value into a character string.
Format	STR$ (Nemerical expression)

The STR$ function converts a numerical value into a character.

Example

What are the execution results of Programs (1) and (2)?

Program (1)

```
10 A=25:B=30
20 C$=STR$(A+B)
30 PRINT C$
40 END
```

Program (2)

```
10 A=25:B=30
20 C$=STR$(A)+STR$(B)
30 PRINT C$
40 END
```

Although these two programs seem to be identical, "55" is displayed for Program (1) and "25 30" is displayed for Program (2).
In Program (1), the result of the numerical expression A+B is converted into a character by the STR$ function. On the other hand, in Program (2), the contents of the numerical variables A and B are converted into respective characters, and are then added. This difference will appear as an execution result difference.

SAMPLE PROGRAM

* Addition practice program.

```
   10 REM ADDITION
   20 FOR I=1 TO 5
   30 X=INT(RND*100)
   40 Y=INT(RND*100)
   50 Z=X+Y
   60 PRINT STR$(X);"+";RIGHT$(STR$(Y),L
EN(STR$(Y))-1);
   70 INPUT "=",A
```

```
 80 IF A=Z THEN PRINT "OK" ELSE 60
 90 NEXT I
100 END
```

(See RIGHT$, LEN.)

This program is an integral addition program for up to two digits. A total of five questions are used. The numeral which is added or the one added to cannot be predicted because the RND function is used.
The STR$ function is used on line 60 where the question is displayed.

Although

```
60 PRINT X; " + ";Y;
```

seems to be reasonable without using STR$, a blank for a + sing occurs before the numerical value as follows when this unnatural system is used.

```
␣15+␣30 = ?
```

This is the difference when a STR$ function is used or is not used.

Reference The VAL function is the reverse of the STR$ function.

LEFT$

Function	Fetches the character string of a character variable from the left by a specified number of characters.
Formats	LEFT$ (Character expression, numerical expression) LEFT$ (“Character string,” numerical expression) LEFT$ (Character variable, numerical expression)

LEFT$ is a function which fetches the character string assigned to the character variable from the left by a specified number of characters.

Example

```
10 DIM AB$
20 AB$=“LEFT RIGHT”
30 B$=LEFT$ (AB$,4)
40 PRINT B$
```

..... Attention should be given to the fact that since the assigned character string is more than 7 characters, a registered variable is used (assigned up to 16 characters).

When it is executed, LEFT is displayed. In other words, four characters from the left of the AB$ character string are fetched.
When 0 is specified as the number of characters to be fetched, a Null is provided, and when the number of characters is specified that exceeds the existing number, an ST error occurs.
The number of characters to be fetched can be specified in the variable or numerical expression.

SAMPLE PROGRAM

* A program which serially increases the character string display.

```
10 AB$="READ LEFT"
20 N=LEN(AB$)
30 FOR I=1 TO N
40 BC$=LEFT$(AB$,I)
50 PRINT BC$
60 NEXT I
70 END
```

Execution Example

```
R
RE
REA
READ
READ
READ L
READ LE
READ LEF
READ LEFT
```

SEE RIGHT$

RIGHT$

Function	Fetches the character string of a character variable from the right by a specified number of characters.
Formats	RIGHT$ (Character string, numerical expression) RIGHT$ (Character variable)

RIGHT$ is a function which fetches a character string assigned to a character variable from the right by a specified number of characters.

Example

```
10  AB$="LEFT RIGHT"
20  B$=RIGHT$(AB$,5)
30  PRINT  B$
```

..... Attention should be given to the fact that since the character string is more than seven characters, a registered variable is used.

When the program is executed, RIGHT is displayed which indicates that five characters from the right of the AB$ character string are fetched. When 0 is specified for the number of characters to be fetched, a null is provided, and when the number of characters is specified that exceeds the existing number, an error occurs.
The number of characters can be specified by a variable or numerical expression.

SAMPLE PROGRAM

* A program which inserts a character string in a character string.

```
10 AB$="AM PM"
20 BC$="NOON"
30 CD$=LEFT$(AB$,2)
40 CD$=CD$+"  "+BC$
50 CD$=CD$+RIGHT$(AB$,3)
60 PRINT CD$
70 END
```

In this program, the BC$ character string is inserted in the AB$ character string by using LEFT$ and RIGHT$.

Execution Result

```
      AM NOON PM
```

SEE LEFT$.

MID$

Function	Fetches a character string that consists of a specified number of characters from a specified position to the right.
Formats	MID$ ("Character string," numerical expression 1, numerical expression 2) MID$ (Character variable, numerical expression 1, numerical expression 2)

MID$ is a function which fetches a character string formed by a specified number of characters starting from the specified position of the specified character string. This function is similar to that of LEFT$ and RIGHT$ together.

$$\text{MID\$}\ (\underset{*}{\underline{\text{A\$}}},\underset{**}{\underline{3}},\underset{***}{\underline{2}})$$

Fetches $\underset{**}{\boxed{\text{the 3rd character from the left}}}$ of the $\underset{*}{\boxed{\text{A\$ character string,}}}$

$\underset{***}{\boxed{\text{by two character positions}}}$.

Example

```
10  CLEAR
20  DIM A$(0)*20
30  A$(0)="LEFT⌴CENTER⌴RIGHT"
40  B$=MID$(A$(0),6,6)
50  PRINT B$
60  END
```

..... Up to 20 characters can be assigned by a defined-length character array. (See Page 171.)

When this program is executed, the A$(0) character string is fetched from the 6th character by 6 character positions, and CENTER is displayed.
Precautions shall be taken for the following items when the MID$ function is used.
When a MID$ (Character expression, n, m) is entered:
(1) The value of n and m is that in which the fractional part is discarded.
(2) When m is 0 and there is no character to be fetched, a null is provided.

(3) When " ,m " is omitted, all the characters from the nth digit and after are provided.
(4) When m exceeds the residual number of characters, all characters from the nth digit and after are provided.
(5) When n is larger than the character length, a null is provided.
(6) Variables and numerical expressions can be used for n and m.
(7) When n and m are outside the range of

$$1 \leqq n < 256 \text{ and } 0 \leqq m < 256,$$

an error (BS error) occurs.

SAMPLE PROGRAM

* A program which counts the number of small " r " alphabetical characters in a composition.

```
10 DIM A$(0)*50
20 N=0
30 INPUT "DATA=";A$(0)
40 M=LEN(A$(0)):N=0
50 FOR I=1 TO M
60 IF MID$(A$(0),I,1)="r" THEN N=N+1
70 NEXT I
80 PRINT N
90 END
```

In this program, an alphabetical statement entry is made, the number of small " r " alphabetical characters is counted, and the number is displayed.
The number of characters that can be entered in an alphabetical statement is up to 50 characters including spaces.
For example, when the following statement is entered;

Learning ⌴ to ⌴ master ⌴ your ⌴ CASIO ⌴ Personal ⌴ Computer

a check is made of each character to see if it is " r " from the first character, and the number of " r " characters is counted.
When the statement shown above is entered, " 5 " is displayed. Try this yourself.

SEE LEFT$, RIGHT$

LEN

Function	Provides the length of a character string.
Formats	LEN "Character string" LEN (Character variable)

LEN is a function that provides the length of a character string assigned by a character variable.

Example

When the following is entered,

CLEAR ↵
PRINT LEN (A$) ↵

"0" is displayed.
This is natural because the A$ variable is emptied by the CLEAR command.

AB$ = "CASIO ⌴ COMPUTER" ↵
PRINT LEN (AB$) ↵

then "14" is displayed.
The range of values provided by the LEN function is 0—79.

SAMPLE PROGRAM

* A character string is displayed with right justification.

```
10 INPUT AB$
20 L=20-LEN(AB$)
30 LOCATE L,3
40 PRINT AB$;:LOCATE 0,0
50 END
```

When character string input is performed in this program, it is displayed with right justification up to the last column of the screen. The number of characters that can be entered for one line is up to 16 characters.

INKEY$

Function	Provides the entry of 1 character from the keyboard.
Format	INKEY$

Although the INKEY$ function is a kind of INPUT command that performs a role similar to that of an INPUT command, it is slightly different.
Data input by INKEY$ function can only be performed when INKEY$ is executed in which key (data) input is performed for one character. If a key is not pressed, the following command is executed with a state in which no input has been performed (null). It is not necessary to press the ⏎ key during data input.
The difference between INKEY$ and INPUT is as shown below.

Use	INKEY$	INPUT
Display during command execution	Nothing is displayed.	(Input request message. It is possible not to make a display of "？ ".)
Data input	Key depression during command execution. (No input without key depression.)	Data entered until ⏎ is pressed.
Kinds of data	1 character as a character.	Digits or number of characters within the range of a variable such as a numerical value, or a character.
Execution of next command	Immediate execution (⏎ depression is unnecessary.)	Suspended until ⏎ is pressed.

Example

Since INKEY$ treats a depressed key as a character, it is generally used in the form of an assignment expression as follows.

Character variable = INKEY$

```
100  A$= INKEY$
110  IF A$=""THEN 100
120  IF A$="E"THEN END
130  ...............................
```

When the "E" key is pressed, this program is terminated, and only when other keys are pressed, it jumps to the execution of the following command. If no key is pressed, it endlessly loops lines 100–110.
In additon, although INKEY$ reads all the keys except the [BRK] key, [SHIFT] key and [CAPS] key are processed as a null.
When the [SHIFT] key or [CAPS] key is pressed with another key at the same time, a character in the SHIFT mode or CAPS mode is provided. However, a one key command provides a null.

SAMPLE PROGRAM

* This is a subroutine which fetches a determined number of characters to a character variable.

```
10 AB$=""
20 FOR I=1 TO 16
30 K$=INKEY$
40 IF K$="" THEN 30
50 IF K$="*" THEN 80
60 AB$=AB$+K$
70 NEXT I
80      ≀
```

Only the number of characters within a determined range can be assigned to a character variable.
If an appropriate character string is entered by an INPUT statement, the number of characters outside the determined range becomes an error. However, in this program, when up to 16 characters have been entered, no input is accepted and the program moves to the following command (line 80).
Also, to stop the input with 16 characters or less, input "*" after which the program moves to the following command (line 80).
Try this by performing the input with

```
80 PRINT AB$
```

for line 80 in the above program.

```
P0    10 A$=INKEY$
      20 IF A$="0" THEN 100
      30 IF A$="1" THEN 200
      40 IF A$="2" THEN 300
      50 GOTO 10
              ⋮
```

Attention should be given to the fact in the above program that if the program is executed by pressing SHIFT P0 and 0 is continuously pressed without SHIFT being pressed, "0" will be read with INKEY$ on line 10.

SEE INPUT

4-5 DISPLAY FUNCTIONS

TAB

Function	Moves the cursor by a specified number of digits.
Format	TAB(Numerical expression) 0 ≦ Numerical expression < 80

This function is used in PRINT and LPRINT statements to move the cursor and the display will be performed at a designated location.

Example

```
10 FOR X=1 TO 5
20 PRINT X;"^2:";
30 PRINT TAB(10);X^2
40 NEXT X
```

When you run this program, the display is as follows.

```
△1^2:␣␣␣␣␣△1
△2^2:␣␣␣␣␣△4
      ⋮
```

TAB(10)

△ indicates the space where the plus sign is omitted.

When TAB (10) is specified as mentioned above, the cursor is moved by 10 display positions, and the following display begins after that.
The range that can be specified by the TAB function is 0—79 including variables and numerical expressions.
A value after the decimal point is discarded.

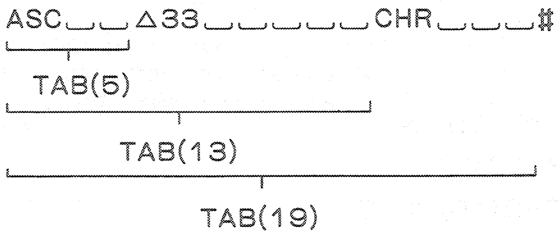
SAMPLE PROGRAM

* Displays a character which corresponds to an ASCII code at a designated location.

```
10 FOR I=33 TO 254
20 PRINT "ASC";
30 PRINT TAB(5);I;
```

```
40 PRINT TAB(13);"CHR";
50 PRINT TAB(19);CHR$(I)
60 NEXT I
70 END
```

When you run this program, the display will be as follows.



```
ASC␣␣△33␣␣␣␣␣CHR␣␣␣#
```

When the ⏎ key is pressed, the following code and character are displayed at the same location.

SAMPLE PROGRAM

* Provides the values of SIN and COS for angles.

```
                                  Excution Result
 10 LPRINT "DEG"                  DEG
 20 LPRINT TAB(10);"SIN";                   SIN            COS
 30 LPRINT TAB(25);"COS"
 40 LPRINT                         0         0              1
 50 FOR X=0 TO 180 STEP 30         30        0.5            0.8660254038
 60 LPRINT X;                      60        0.8660254038   0.5
 70 LPRINT TAB(10),SINX;           90        1              0
 80 LPRINT TAB(25),COSX            120       0.8660254038  -0.5
 90 NEXT X                         150       0.5           -0.8660254038
100 END                            180       0             -1
```

SEE USING, PRINT, LPRINT, LOCATE

USING

Function	Specifies a display format.
Format	USING "Format character string";

The USING function displays a numerical value or a character string in a PRINT or LPRINT statement by using a certain specified uniform format.

Example

When numerical values are displayed in several lines, sometimes digit or decimal point deviations occur. However they can be arranged properly by utilizing the USING statement as follows.

```
10  A=18.5
20  B=2.67
30  C=135.78
40  PRINT USING  "#####.###" ;A
50  PRINT USING  "#####.###" ;B
60  PRINT USING  "#####.###" ;C
70  END
```

When you execute this program, the display can be easily read as follows.

```
 ␣␣␣18.500
␣␣␣␣2.670
 ␣␣135.780
```

" # " and " . " as used here are the format character string.
Also, they can be used for a character string display. The format for a character string uses &.

```
10  AB$= "CASIO␣COMPUTER" : B$= "!!"
20  PRINT USING "&&&&&&&&&&&&&&&" ; AB$;B$
30  END
```

When you execute this program, the display will be as follows.

CASIO␣COMPUTER␣␣!!

The number of characters for the CASIO␣COMPUTER is 14, however since a 16 format character string is used, the two extra characters are displayed as blanks.

When a USING statement is used, the following precautions shall be taken.

(1) Characters other than # , · , ^ and & cannot be used in a character string format.

(2) # , · , ^ and & cannot be used together.

(3) Numerical format specification

...... Numerical digit specification
· ...... Decimal point specification
^ ...... Exponent specification

(a) # can be specified within 13 digits before the decimal point and within 9 digits after the decimal point, and altogether within 13 digits.

$$\underbrace{\#\#\#\#\#\#\#\#\#\#\#\#\#}_{13}. \qquad .\overbrace{\#\#\#\#\#\#\#\#\#}^{9}$$

$$\underbrace{\#\#\#\#\#\#\#\#.\#\#\#\#\#}_{13}$$

(b) Specified by including a minus sign in #

(c) ^ ^ ^ ^ is always specified without any relationship to how many ^ are used.

(d) When the fractional portion exceeds the format, output is performed by rounding the next numerical value of a specified digit to the nearest whole number.
PRINT USING "##.##"; 1234.56 → %␣1234.56

(e) When the integral portion exceeds the format, % is placed at the beginning to allow output without following the format.
PRINT USING "###.###"; 12.3456 → 12.346

(f) Numerical value output is performed with right justification.

(4) & character string format specification

(a) & can be written as much as desired.

(b) If the number of & is smaller than the character string, output is performed from the beginning by the number of & positions.
PRINT USING "&&&&"; "ABCDEF" → ABCD

(c) Character string output is performed with left justification.

(d) If the number of & is larger than the character string, the output of spaces is performed.

(5) The USING specification is only effective in one PRINT or LPRINT statement.

(6) A USING specification is renewed by a new USING specification.

(7) A USING specification can be released by USING " ";.

SAMPLE PROGRAM

* A program which displays a space between each character of a character string.

```
10 A$="CASIO"
20 PRINT A$
30 FOR I=1 TO LEN(A$)
40 M$=MID$(A$,I,1)
50 PRINT USING"&&&";M$;
60 NEXT I
70 PRINT
80 END
```

When you run this program,

C␣␣A␣␣S␣␣I␣␣O is displayed.

Each character in the character string is fetched by the MID$ function which is displayed by the USING function format.

SAMPLE PROGRAM

* A program which outputs a name, the height, and weight to the printer with a certain format.

```
 1100 REM ***USING***
 1110 CLEAR :DIM A$(1),A!(1),B!(1)
 1120 FOR I=0 TO 1
 1130 INPUT "NAME";A$(I),"HEIGHT(cm)";A!
(I),"WEIGHT(kg)";B!(I)
 1140 LPRINT :NEXT I
 1150 FOR I=0 TO 1
 1160 LPRINT TAB(2);USING"&&&&&&&&&&&&&&
&";A$(I);
 1170 LPRINT USING"######";A!(I);"cm";B!
(I);"Kg"
 1180 NEXT I
 1190 END
```

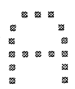
Execution Example

```
  JOHN SMITH          190cm    185Kg
  BOB JONES            68cm      7Kg
```

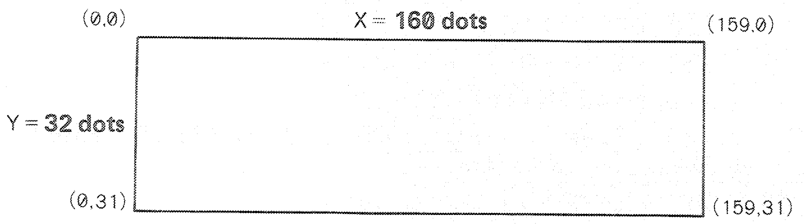
POINT

Function	Checks if a display dot is lit or not.
Format	POINT (X, Y) $0 \leqq X \leqq 159$ (Horizontal position) $0 \leqq Y \leqq 31$ (Vertical position)

It can be determined that a character or sign consists of small square dots on the display screen.
For example, the character A is displayed as follows.



Each point is called a dot.
The entire display consists of 5120 dots.



By assuming that the horizontal direction is the X axis and that the vertical direction is the Y axis, one display dot can be displayed by using

Coordinate (X, Y)

The POINT function checks if a dot (X, Y) is lit or not.
When a dot is lit on the coordinates (X, Y), "1" is used, and when it is turned off, "0" is used.

Example

```
10  X=0:Y=0
20  A=POINT (X,Y)
30  PRINT  A
```

When you execute this program, if the (0, 0) dot is lit, "1" is displayed, and if it is turned off, "0" is displayed.
Precautions shall be taken for the following items when the POINT function is used.

(1) Values rounded at 1 decimal place are used for X and Y.
(2) When X and Y exceed the range of the coordinates, an error (BS error) occurs.

SAMPLE PROGRAM

* Laser gun program.

```
    100 CLS
    110 X=INT(RND*10)+75
    120 DRAW(X,0):DRAW(80,29)-(80,31)
    130 N$=INKEY$
    140 IF N$=" " THEN GOSUB 300
    150 DRAWC(X,0)
    160 GOTO 110
    300 DRAW(80,28)-(80,1)
    310 A=POINT(80,0)
    320 IF A=1 THEN LOCATE 0,3:PRINT "BEE!
":BEEP
    330 DRAWC(80,28)-(80,1)
    340 FOR I=0 TO 30:NEXT I
    350 CLS :RETURN
```

CHAPTER 5

PROGRAM LIBRARY

PLEASE NOTE:

Programs in this chapter may be used freely without permission. However, it must be understood that the company is not responsible for any damage or loss as a result of using these examples.

In the case of executing programs in this chapter without the optional plotter-printer (FA-10), press "N" when the PB-700 requests whether printouts are made or not by displaying "PRINTER ON? (Y/N)".

STOCK PRICE MANAGEMENT AND PROPER SELLING/BUYING PRICES

This program stores stock prices for the past 53 weeks. Each time data of the new week is entered when data of 53 weeks have been stored, the data of the oldest week is discarded. Based on the stock price data, the program outputs the current deviation value and helpful information for judgment on buying or selling. The program also permits displaying the combination of deviation value and moving average and graphically displaying stock price fluctuations.

Explanation

First, start the program P0, and the menu is displayed on the screen. Then, enter the appropriate number from 1 to 7 given with the menu. Entering a number other than 1 to 7 causes the menu to be displayed again.

(1) Data input

To input data, press 1 after the menu is displayed, and "Initial? (Y/N)" is displayed. If you input data for the first time, press the Y key. If you input data following another data, press the N key. When the Y key is pressed, "CLEAR OK? (Y/N)" is displayed on the screen. This is to prevent data from being lost by erroneous input. Normally, press the Y key.
For the initial input, "1) DATA=" is displayed on the screen. Enter the appropriate stock price and press the ↵ key. For the second and subsequent data inputs, the screen displays "WEEK=". For input of data for the 54th and subsequent weeks, the oldest data is sequentially erased, therefore the time required for input becomes a little longer. To terminate data input, enter a negative number, and the menu is displayed again.

(2) Judgment on sell or buy

When menu 2 is selected, "Current Price? " is displayed. At this time, enter the current stock price, and the program compares the current stock price with the past data and outputs the deviation value. To jump out from this routine, enter a negative value, and the menu is displayed again.

(3) Checking reasonable stock price

When executing this routine, you should remember one thing: if new data has been entered, the routine (2) must be executed before this routine so that the correct value can be output. This is because the two routines share part of the same variables. Instead, this routine permits immediate data input. Entering a negative number causes the menu to be displayed again.

(4) Data output

When this menu is selected, all the data stored in memories are displayed on the screen, then the menu appears again.

(5) Moving average

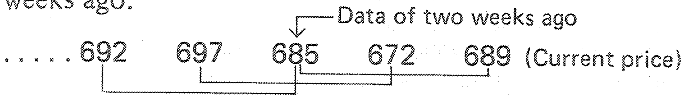
This routine calculates the moving average. When "No. of movements? " is displayed, enter the number of weeks for which the moving average is to be calculated, and the routine calculates the moving average of the period between the current week and the specified week. After this routine is executed, the menu is automatically displayed.

(6) Moving average in the past

This routine permits reviewing the change in moving average in the past, so this can be used to determine whether the stock price is rising up or falling down.
When "No. of movements? " is displayed, enter the number of weeks for which the moving average is to be calculated. Then, "FROM WHEN? " is displayed. Here, enter the week from which the number of weeks is to be counted.

Example:

Calculating the moving average for each three weeks, starting from two weeks ago.

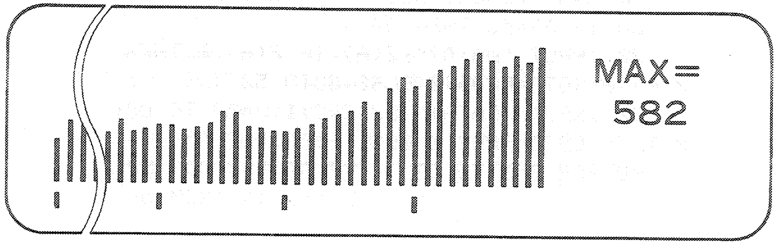


The menu automatically appears after executing this routine. In regard to routines (5) and (6), use care when entering data in order to prevent an incorrect value from being output, especially when the amount of data (the number of weeks) stored is relatively small.

(7) Graph

This routine graphically displays the stored data to permit easy recognition of the general trend of the stock price.
The menu display appears again after executing this routine.

Execution Example of a Graph Display



Program

```
 P0
   10 CLS
   20 GOSUB 80
   30 INPUT "INPUT    NO.";PR
   40 IF PR>7 THEN 10 ELSE IF PR<1 THEN
10
   50 IF PR=1 THEN GOTO PROG 1 ELSE IF P
R=2 THEN GOTO PROG 2
   60 IF PR=3 THEN GOTO PROG 3 ELSE IF P
R=4 THEN GOTO PROG 4
   70 IF PR=5 THEN GOTO PROG 5 ELSE IF P
R=6 THEN GOTO PROG 6 ELSE GOTO PROG 7
   80 PRINT :PRINT "DATA INPUT 1"
   90 FOR I=1 TO 100:NEXT I
  100 PRINT "PRICE CHECK 2"
  110 FOR I=1 TO 100:NEXT I
  120 PRINT "REASONABLE PRICE 3"
  130 FOR I=1 TO 100:NEXT I
  140 PRINT "DATA OUTPUT 4"
  150 FOR I=1 TO 100:NEXT I
  160 PRINT "MOVING AVE. 5"
  170 FOR I=1 TO 100:NEXT I
  180 PRINT "PAST MOVEMENT 6"
  190 FOR I=1 TO 100:NEXT I
  200 PRINT "GRAPH DISPLAY 7"
  210 RETURN
 P1
   10 CLS
   20 PRINT " ** DATA INPUT **"
   30 INPUT "Initial?(Y/N)",P$:IF P$="Y"
 THEN GOSUB 200
   40 IF P$="N" THEN A=A-1 ELSE IF P$<>"
Y" THEN 30
   50 A=A+1:GOSUB 300
   60 IF A>=53 THEN 80
   70 INPUT "DATA?",Z(A):IF Z(A)<0 THEN
Z(A)=0:GOTO PROG 0 ELSE GOTO 50
   80 INPUT "DATA+",DZ:C=C+1:A=53:IF DZ<
0 THEN GOTO PROG 0 ELSE Z(53)=DZ
   90 FOR B=1 TO 53
```

```
  100 Z(B-1)=Z(B)
  110 NEXT B
  120 GOTO 80
  200 INPUT "CLEAR OK?(Y/N)",T$:IF T$="Y
" THEN 220 ELSE IF T$="N" THEN 260
  210 GOTO 200
  220 ERASE :DIM Z(53)
  230 PRINT "STOCK PRICE "
  240 FOR K=0 TO 50:NEXT K
  250 INPUT "1)DATA=",Z(1):A=1
  260 P$="Y":RETURN
  300 PRINT "WEEK=";A
  310 FOR K=0 TO 10:NEXT K:RETURN
 P2
   10 CLS
   20 PRINT " ** PRICE CHECK **"
   30 S=0:Q=0
   40 FOR D=0 TO 52
   50 S=S+Z(D):Q=Q+Z(D)^2
   60 NEXT D
   70 IF A<54 THEN 90
   80 E=S/53:V=Q-53*E*E:F=SQR(V/52)
   90 E=S/A:V=Q-A*E*E:F=SQR(V/(A-1))
  100 INPUT "Current Price";Y:IF Y<0 THE
N 140
  110 D=ROUND(50+10*(Y-E)/F,-3)
  120 PRINT "Deviation=";D
  130 GOTO 100
  140 GOTO PROG 0
 P3
   10 CLS
   20 PRINT "**REASONABLE PRICE**"
   30 INPUT "Deviation=";D:Y=ROUND((D-50
)*F/10+E,-2):IF D<0 THEN 50
   40 PRINT " PRICE=";Y:GOTO 30
   50 GOTO PROG 0
 P4
   10 CLS
   20 PRINT " ** DATA OUTPUT **"
   30 FOR U=0 TO A-2
   40 PRINT "DATA";U+1;"=";Z(U+1)
   50 FOR K=0 TO 50
   60 NEXT K:NEXT U
  70 PRINT "DATA END"
  80 FOR K=1 TO 200:NEXT K
  90 GOTO PROG 0
P5
  10 CLS
  20 PRINT " ** MOVING AVE. **"
  30 X=0
  40 INPUT "No. of movements";N
  50 IF A<=53 THEN 100
  60 FOR L=53-N TO 52
  70 X=X+Z(L)
  80 NEXT L
  90 GOTO 130
 100 FOR K=A-N TO A
 110 X=X+Z(K)
 120 NEXT K
 130 M=X/N
 140 PRINT "MOVING AVERAGE";M
 150 FOR K=0 TO 300:NEXT K
 160 GOTO PROG 0
P6
  10 CLS
  20 PRINT "** PAST MOVEMENT **"
  30 INPUT "No. of movements";I
  40 INPUT "FROM WHEN ";O
  50 X=0
  60 IF A<=53 THEN 110
  70 FOR J=53-O-I TO A-O-I
  80 X=X+Z(J)
  90 NEXT J
 100 GOTO 140
 110 FOR J=A-O-I TO A-O-1:IF A<=J THEN
180
 120 X=X+Z(J)
 130 NEXT J
 140 M=X/I
 150 FOR K=0 TO 100:NEXT K
 160 PRINT "Movins Ave.";M
 170 O=O-1:X=0:FOR K=0 TO 10:NEXT K:GOT
O 60
 180 PRINT " END"
 190 FOR K=0 TO 300:NEXT K
 200 GOTO PROG 0
```

```
P7
  10 CLS
  20 PRINT "** GRAPH DISPLAY **"
  30 MX=0:FOR MD=1 TO 53
  40 IF Z(MD)>MX THEN MX=Z(MD)
  50 NEXT MD
  60 LOCATE 15,2:PRINT "MAX=":LOCATE 15
,3:PRINT MX
  70 FOR K=1 TO A
  80 J1=K*2+10:J2=25-25/MX*Z(K)
  90 FOR K=1 TO 5:PO=53*2+10-K*20+2
 100 DRAW(PO,27)-(PO,30):NEXT K
 110 FOR K=1 TO A
 120 J1=K*2+10:J2=25-25/MX*Z(K)
 130 DRAW(J1,J2)-(J1,25):NEXT K
 140 FOR K=0 TO 300:NEXT K
 150 GOTO PROG 0
```

Example Data

19 weeks ago	584	14 weeks ago	545	9 weeks ago	635	4 weeks ago	685
18 weeks ago	580	13 weeks ago	550	8 weeks ago	652	3 weeks ago	697
17 weeks ago	579	12 weeks ago	563	7 weeks ago	673	2 weeks ago	685
16 weeks ago	570	11 weeks ago	589	6 weeks ago	701	Last week	672
15 weeks ago	562	10 weeks ago	620	5 weeks ago	692	This week	689

Variable contents

A	Data counter	J 2	Y axis of graph	Q	Sum of squares of data
B ~ D	Counters	MD	Counter		
DA	Character data	MX	Max. data	S	Sum of data
DZ	Stock price	N	No. of movements	T$	CLEAR OK?
E	Average	O	Starting week for calculating of moving average	U	Counter
H	Deviation value			V	Variance
J ~ L	Counters (for periods to calculate moving average in P5)			X	Counter
		P$	Y or N (Initial?)	Z (X)	Stock price
		P0	Graph scale		
J 1	X axis of graph	PR	Program selection		

Operation

	Step	Key operation	Display
Menu display		SHIFT P0	DATA INPUT 1 PRICE CHECK 2 REASONABLE PRICE 3 DATA OUTPUT 4 MOVING AVE. 5 PAST MOVEMENT 6 GRAPH DISPLAY 7 INPUT NO. ? _
Selects the menu of data input.	1	1 ↵	**DATA INPUT** Initial ? (Y/N)_
Press Y for initial data input and N for additional data input.	2	Y ↵	CLEAR OK ? (Y/N)_
Press Y to clear the stored data and N not to clear them.	3	Y ↵	STOCK PRICE 1) DATA ? _
(1) Data input	4	584 ↵ ⋮	WEEK=2 DATA ? _ ⋮
Enter a negative number to terminate data input.	5	−1 ↵	≀ INPUT NO. ? _
(2) Stock price check	6	2 ↵	**PRICE CHECK** Current Price ? _
Enter the current stock price and its deviation value is displayed based on the past data.	7	585 ↵	Deviation =49.23 Current Price ? _
Entering a negative number causes the menu to be displayed.	8	−1 ↵	≀ INPUT NO. ? _
(3) Reasonable stock price	9	3 ↵	**REASONABLE PRICE** Deviation= ? _
Enter a deviation value. The stock price is displayed.	10	55 ↵	PRICE=670.1 Deviation= ? _
Entering a negative number causes the menu to be displayed.	11	−1 ↵	≀ INPUT NO. ? _
(4) Data output All data are displayed and then the menu display appears. Menu display	12	4 ↵	**DATA OUTPUT** DATA 1=584 DATA 2=580 DATA 3=579 ⋮ DATA END ≀ INPUT NO. ? _
(5) Moving average	13	5 ↵	**MOVING AVE.** No. of movements ? _
Enter the number of movements (weeks).	14	15 ↵	MOVING AVERAGE 643.2 ≀ INPUT NO. ? _
(6) Moving average in the past.	15	6 ↵	**PAST MOVEMENT** No. of movements ? _
Enter the number of movements (weeks) for which the moving average is to be calculated.	16	3 ↵	FROM WHEN ? _
Enter the starting week. Each moving average is displayed and then the menu display appears.	17	2 ↵	Moving Ave. 689 Moving Ave. 684.666·············· Moving Ave. 682 END ≀ INPUT NO. ? _
(7) Graph	18	7 ↵	A graph is displayed.

TELEPHONE DIRECTORY

This program permits immediately recalling a desired telephone number by entering previously stored names. It also permits recalling a phone number using the initial letter of a name. Names can also be arranged in alphabetical order.

Explanation

By storing the phone numbers of your friends and acquaintances, you can immediately recall the desired phone number by this program. Up to 50 names can be stored at a time. Once the names and phone numbers have been stored, it is possible to recall the desired phone number merely by entering the initial letter or the first few letters of the name. First, execute the program P0, and the menu is displayed on the screen. Enter the appropriate number, 1 to 4. There is no need to press the ↵ key.

(1) INPUT

Stores names and phone numbers. Input data by the following procedure. Data to be input is indicated by an underscore.

NAME? CASIO ↵
TEL NO.? 123-4567 ↵

Input all names and phone numbers by repeating the above procedure. When the last name and phone number are entered, enter END ↵. The screen displays the menu again.

(2) SORTING

Arranges the stored names in alphabetical order. While the names are being sorted, 'SORTING . . . ' is displayed on the screen. The sort operation is completed in several seconds to a few minutes depending on the amount of data stored. The sorted names (and the associated phone numbers) can be sequentially displayed on the screen by pressing any key on the keyboard. After all the names and phone numbers are displayed, the menu is displayed again.

(3) LOOK FOR

Recalls the phone number by entering a name.
Enter the name as follows:

NAME? CASIO ↵

And the name and phone number are displayed as follows:

CASIO
123-4567

Note that only the initial letter or the first few letters of the name may be entered to recall the phone number. In this case, if there is more than one name whose initial letter or first few letters are the same, all of them are displayed. Note also that if identical names are stored, they are all displayed. If any name which has not been stored is entered, 'NO DATA' is displayed on the screen. In this case, press any key on the keyboard to return to the menu.

(4) DELETE

Deletes data which has been stored.
Enter the name to be deleted as follows:

NAME? ABCDE ↵

Then, the screen displays:

ABCDE
XXX-XXXX Y/N? (XXX-XXXX: phone number of ABCDE)

If you really wish to delete the name and phone number, press the Y key. If not, press the N key. This is to assure you that no data is erroneously deleted.
To clear all the data which have been stored, execute the CLEAR command. Since menu 1 (INPUT) has the function to add data, it may be combined with menu 4 (DELETE) to add or delete data freely.

Program

```
P0
 10 CLS
 20 PRINT "1-INPUT   2-SORTING"
 30 PRINT "3-LOOK FOR"
 40 PRINT "4-DELETE"
 50 K$=INKEY$:IF K$="" THEN 50
 60 IF K$="1" THEN GOTO PROG 1
 70 IF K$="2" THEN GOTO PROG 2
 80 IF K$="3" THEN GOTO PROG 3
 90 IF K$="4" THEN GOTO PROG 4
100 GOTO 50
P1
 10 CLS
 20 N=N+1
 30 IF N=51 THEN 90
 40 IF N=1 THEN DIM A$(50),B$(50)*12
 50 INPUT "NAME   ";A$(N)
 60 IF A$(N)="END" THEN 100
 70 INPUT "TEL NO.";B$(N)
 80 GOTO 10
 90 PRINT "FULL";BEEP 1
100 N=N-1
110 GOTO PROG 0
P2
  5 CLS :PRINT "SORTING..."
 10 FOR I=1 TO N
 20 MM$=A$(I):X=I
 30 FOR J=I TO N
 40 KK$=A$(J)
 50 GOSUB 200
 60 NEXT J
 70 A$(X)=A$(I):A$(I)=MM$
 80 MM$=B$(X)
 90 B$(X)=B$(I):B$(I)=MM$
100 NEXT I
110 GOTO PROG 5
200 KU=0
210 KU=KU+1
230 O1=LEN(MM$):O2=LEN(KK$)
240 IF KU>O1 THEN RETURN  ELSE IF KU>O
    2 THEN 300
250 MI$=MID$(MM$,KU,1):KI$=MID$(KK$,KU
    ,1)
260 IF ASC(MI$)=ASC(KI$) THEN 210
270 IF ASC(MI$)>ASC(KI$) THEN X=J:MM$=
    KK$:RETURN
280 RETURN
300 X=J:MM$=KK$:RETURN
P3
 10 CLS
 20 INPUT "NAME   ";MM$
 30 X=LEN(MM$)
 40 I=0
 50 I=I+1
 60 IF I=N+1 THEN IF F=1 THEN 120 ELSE
     130
 70 IF MM$=LEFT$(A$(I),X) THEN 90
 80 GOTO 50
 90 F=1:PRINT A$(I)
100 PRINT B$(I)
```

```
110 K$=INKEY$:IF K$="" THEN 110 ELSE 5
    0
120 F=0:GOTO PROG 0
130 PRINT "NO DATA"
140 K$=INKEY$:IF K$="" THEN 140 ELSE 1
    20
P4
 10 CLS
 20 INPUT "NAME   ";MM$
 30 X=LEN(MM$)
 40 I=0
 50 I=I+1
 60 IF I=N+1 THEN 180
 70 IF MM$=LEFT$(A$(I),X) THEN 100
 90 GOTO 50
100 PRINT A$(I)
110 PRINT B$(I);" Y/N ?"
120 K$=INKEY$:IF K$="" THEN 120
130 IF K$="Y" THEN 150
140 GOTO 50
150 A$(I)=A$(N)
160 B$(I)=B$(N)
170 N=N-1
180 GOTO PROG 0
P5
 10 FOR I=1 TO N
 20 CLS
 30 PRINT A$(I)
 40 PRINT B$(I)
 50 K$=INKEY$:IF K$="" THEN 50
 60 NEXT I
 70 GOTO PROG 0
```

Operation

	Step	Key operation	Display
Menu display		SHIFT PO 0	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(1) INPUT Select menu 1	1	1	NAME ?_
Input the first person's name.	2	SMITH, JOHN ⏎	NAME ? SMITH, JOHN TEL NO. ?_
Input his telephone number.	3	03-583-4111 ⏎	NAME ?_
Input the second person's name.	4	BROWN, MARY ⏎	NAME ? BROWN, MARY TEL NO. ?_
Input her telephone number.	5	052-264-1453 ⏎	NAME ?_ ⋮
After completing data input, enter END ⏎ and then the menu is displayed on the screen.	6	END ⏎	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(3) LOOK FOR Whose telephone number do you want to know?	7	3	NAME ?_
Only a family name can be entered.	8	SMITH, JOHN ⏎	NAME ? SMITH, JOHN SMITH, JOHN 03-583-4111
The menu is displayed on the screen.	9	⏎	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(2) SORTING Alphabetical order	10	2	SORTING...
After "SORTING..." disappears	11	⏎	ALLEN, ROBERT 06-314-2681
The sorted names can be sequentially displayed.	12	⏎	BROWN, MARY 052-264-1453 ⋮

	Step	Key operation	Display
After all the names and phone numbers are displayed, the menu is displayed.	13	⏎	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
(4) DELETE What name do you want to delete?	14	4	NAME ?_
If you really want to delete, press Y. If not, press N.	15	SMITH, JOHN ⏎	NAME ? SMITH, JOHN SMITH, JOHN 03-583-4111 Y/N ?_
The menu is displayed.	16	Y	1—INPUT 2—SORTING 3—LOOK FOR 4—DELETE
Confirm that the data is deleted by using menu 3.	17	3	NAME ?_
"NO DATA" indicates that the name has not been stored.	18	SMITH, JOHN ⏎	NAME ? SMITH, JOHN NO DATA

Variable contents	
N	(Number of NAMEs/TEL NOs)−1
X	Length of character string to be looked for
A$(0,0) ≀ A$(50,0)	Names
A$(0,1) ≀ A$(50,1)	Telephone numbers
MM$	Character string to be looked for

CROSS TOTAL

This program permits obtaining sums of horizontal (X) and vertical (Y) data, or sorting the data to see the percentage of each data element. For example, the item X may be a certain product and the item Y may be a certain month.

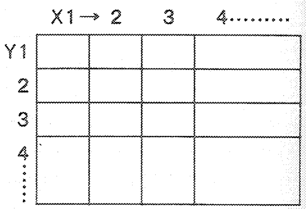
Explanation

First, execute the program P0, and the following menu is displayed on the screen:

```
1  DATA INPUT       ←For inputting data
2  TOTAL            ←For obtaining horizontal and vertical totals
3  SORT             ←For arranging data
4  DATA OUTPUT ?_   ←For checking all the data
```

(1) Data input

When menu 1 (DATA INPUT) is selected, "CLEAR (Y/N)? " is displayed. To input data for the first time, press Y. Then, the program requests you to input the values of X and Y (the value of X is the number of data elements in horizontal direction, and the value of Y is the number of data elements in vertical direction).
Input the appropriate values.
After the values of X and Y have been input, you are requested to input the data. Input the data according to the element numbers displayed on the screen:



X = 1, Y = 1 → X = 2, Y = 1 → X = 3, Y = 1 . . .
X = 1, Y = 2 → X = 2, Y = 2 → X = 3, Y = 2 . . .

After all data are input, the grand total is displayed and the menu display appears.
If the data entered contains an error, enter 1 again. When "CLEAR (Y/N)? " is displayed, enter N. Then, the program asks you about the element number whose associated data is to be corrected. Input the appropriate element number, and "DATA? " is displayed. Input the correct data. The correct grand total is displayed and the menu display appears.

(2) Sum of data in item X or Y

When menu 2 (TOTAL) is selected, "PRINTER ON? (Y/N)" is displayed. The subtotals are printed by entering Y. The program asks you whether you wish to obtain the sum of item X or item Y. Input X or Y, whichever is appropriate.
When X is input, the program outputs the subtotals of X from 1 to the preset value, and returns to the menu display after outputting the grand total. When Y is entered, the program outputs the subtotals of Y from 1 to the preset value, and returns to the menu display after outputting the grand total.

(3) Sorting

When menu 3 (SORT) is selected, the program asks you whether the subtotals of X or Y are to be sorted after displaying "PRINTER ON? (Y/N)". When X or Y is input, "SORTING NOW" is displayed and a sort operation is started. The data sorted in descending order is output, together with the ranking, item name, subtotal and percentage of each element, and then the menu is displayed. The sort operation requires some time. For example, it takes approximately one minute and 10 seconds to sort 20 data elements. When the sort operation is completed, a buzzer sounds and the data output begins. After execution of this program, the menu display appears again.
Since this program uses many half-precision variables, it can handle a relatively large volume of data. Note, however, that the maximum number of digits of input data is five.
To review the result of sorting, press the BRK key and run line 140 of the program P4.

(4) Data output

When menu 4 (DATA OUTPUT) is selected, data such as "X=1 Y=1 DATA=233" is displayed after displaying "PRINTER ON? (Y/N)". After all the data are displayed, the menu display appears again.

Program

```
 P0
   10 PRINT "1 DATA INPUT ","2 TOTAL","3
 SORT","4 DATA OUTPUT ";
   20 INPUT R
   30 IF R=1 THEN GOTO PROG 1
   40 INPUT "PRINTER ON?(Y/N)",F$
   45 IF R=2 THEN GOTO PROG 2
   50 IF R=3 THEN GOTO PROG 3 ELSE IF R=
4 THEN GOTO PROG 5
 P1
   10 INPUT "CLEAR (Y/N)?",S$
   20 IF S$="Y" THEN CLEAR :GOTO 30 ELSE
 IF S$<>"N" THEN 10 ELSE 110
   30 INPUT "X";X,"Y";Y
   40 DIM D!(X,Y),X!(X),Y!(Y)
   50 FOR J=1 TO Y:Y!(J)=0:FOR I=1 TO X
   60 PRINT "INPUT DATA X=";I;" Y=";J,
   70 INPUT D!(I,J)
   80 Y!(J)=Y!(J)+D!(I,J)
   90 NEXT I:NEXT J
  100 GOSUB 200:GOTO PROG 0
  110 PRINT "CORRECTION(X,Y)";
  120 INPUT I,J
  130 INPUT "DATA";D!(I,J)
  140 FOR J=1 TO Y:Y!(J)=0:FOR I=1 TO X
  150 Y!(J)=Y!(J)+D!(I,J)
  160 NEXT I:NEXT J
  170 GOSUB 200:GOTO PROG 0
  200 S=0:FOR I=1 TO X:X!(I)=0:FOR J=1 T
O Y
  210 X!(I)=X!(I)+D!(I,J):S=S+D!(I,J)
  220 NEXT J:NEXT I
  230 PRINT "GRAND T.";S:FOR K=0 TO 100:
NEXT K
  240 RETURN
 P2
   10 INPUT "X-SUM OR Y-SUM";P$
   20 IF P$="Y" THEN GOTO 160 ELSE IF P$
="X" THEN 80 ELSE 10
```

```
   80 FOR K=1 TO X
   90 PRINT "X=";K;" SUM=";X!(K)
   95 IF F$="Y" THEN GOSUB 300
   97 IF INKEY$="" THEN 97 ELSE 100
  100 NEXT K
  110 PRINT "GRAND T.=";S
  115 IF F$="Y" THEN GOSUB 340
  120 IF INKEY$="" THEN 120 ELSE 130
  130 GOTO PROG 0
  160 FOR K=1 TO Y
  170 PRINT "Y=";K;" SUM=";Y!(K)
  173 IF INKEY$="" THEN 173 ELSE 175
  175 IF F$="Y" THEN GOSUB 320
  180 NEXT K
  190 PRINT "GRAND T.=";S
  195 IF F$="Y" THEN GOSUB 340
  200 IF INKEY$="" THEN 200 ELSE 210
  210 GOTO PROG 0
  300 LPRINT "X=";K;" SUM=";X!(K):RETURN

  320 LPRINT "Y=";K;" SUM=";Y!(K):RETURN

  340 LPRINT "GRAND T.";S
  350 GOTO PROG 0
 P3
   10 INPUT "SORT X OR Y?",P$
   20 IF P$="Y" THEN GOSUB 100 ELSE IF P
$="X" THEN GOSUB 200 ELSE 10
   30 GOTO PROG 4
  100 ERASE A!
  110 DIM A!(Y,2)
  120 FOR J=1 TO Y
  130 A!(J,1)=Y!(J):A!(J,2)=J
  140 NEXT J
  150 N=Y:RETURN
  200 ERASE A!
  210 DIM A!(X,2)
  230 FOR I=1 TO X
  240 A!(I,1)=X!(I):A!(I,2)=I:NEXT I
  250 N=X:RETURN
```

```
P4
  10 CLS
  20 PRINT "SORTING NOW"
  30 REM SORT
  40 FOR K=N-1 TO 1 STEP -1
  50 FOR L=1 TO K
  60 IF A!(L,1)>A!(L+1,1) THEN 100
  65 FOR M=1 TO 2
  70 T=A!(L,M)
  80 A!(L,M)=A!(L+1,M)
  90 A!(L+1,M)=T
  95 NEXT M
 100 NEXT L
 110 NEXT K
 120 REM PRINT
 130 FOR K=1 TO 10:BEEP :NEXT K:CLS
 140 FOR K=1 TO N:GOSUB 220
 150 PRINT USING"##";K;" ";P$;"=";USING
"##";A!(K,2);USING"########";A!(K,1);
 160 PRINT USING"###";A;CHR$(37)
 170 IF F$="Y" THEN GOSUB 300:NEXT K:GO
SUB 320:GOTO PROG 0
 180 FOR L=1 TO 50:NEXT L:NEXT K
 190 PRINT "GRAND T.";S
 200 FOR K=0 TO 100:NEXT K
 210 GOTO PROG 0
 220 REM RATIO
 230 A=ROUND(A!(K,1)/S,-3)*100
 240 RETURN
 300 LPRINT USING"##";K;" ";P$;"=";USIN
G"##";A!(K,2);USING"########";A!(K,1);
 310 LPRINT USING"###";A;CHR$(37):RETUR
N
 320 PRINT "GRAND T.";S
 330 LPRINT "GRAND T.";S
 340 RETURN
P5
   5 CLS
  10 FOR J=1 TO Y:FOR I=1 TO X
  20 PRINT "X=";I;" Y=";J;" DATA=";D!(I
,J)
  30 IF F$="Y" THEN GOTO 50
  40 IF INKEY$="" THEN 40 ELSE 60
  50 LPRINT "X=";I;"  Y=";J;" DATA=";D!
```

```
(I,J)
  60 NEXT I:NEXT J:GOTO 80
  70 LPRINT :LPRINT :LPRINT
  80 GOTO PROG 0
```

Variable contents					
A	Percentage.	I, J	Array subscripts.	S$	Y or N.
A!( )	For storing data during sorting.	K~M	Counters.	T	Variable for data exchange.
		N	Number of data elements during sorting.	X!( )	Array for sums of X.
D!( )	Data array.			Y!( )	Array for sums of Y.
F$	Determines whether the printer is used or not.	P$	X or Y.		
		R	Menu selection.		
		S	Grand total.		

* If you wish to handle data more than 5 digits, change variables A!( ), D!( ), X!( ) and Y!( ) as follows: A( ), D( ), X( ) and Y( )

Sample Data

```
X= 1  Y= 1  DATA= 321
X= 2  Y= 1  DATA= 369
X= 3  Y= 1  DATA= 357
X= 1  Y= 2  DATA= 159
X= 2  Y= 2  DATA= 147
X= 3  Y= 2  DATA= 123
X= 1  Y= 3  DATA= 842
X= 2  Y= 3  DATA= 862
X= 3  Y= 3  DATA= 579
```

Operation

	Step	Key operation	Display
The menu display.	1	SHIFT PO 0	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
(1) Data input Do you clear the stored data?	2	1 ↵	CLEAR(Y/N)?_
How many are the horizontal items?	3	Y ↵	X?_

	Step	Key operation	Display
How many are the vertical items?	4	3 ↵	Y ?_
Input data (X=1, Y=1).	5	3 ↵	INPUT DATA X= 1 Y= 1 ?_
Input data (X=2, Y=1).	6	321 ↵ ⋮ Input all the data by repeating above.	INPUT DATA X= 2 Y= 1 ?_
Grand total display			GRAND T. 3759
The menu display.	7		1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
(2) Sum of data in item X Is the printer used? Enter X ↵ for X subtotals and Y ↵ for Y subtotals.	8	2 ↵ N ↵	PRINTER ON ? (Y/N)_ X−SUM OR Y−SUM ?_
Output of each subtotal (SUM) and grand total.	9	X ↵ ↵ ↵ ↵	X=1 SUM=1322 X=2 SUM=1378 X=3 SUM=1059 GRAND T. =3759
The menu display	10	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
(2) Sum of data in item Y Is the printer used?	11	2 ↵ N ↵	PRINTER ON ? (Y/N)_ X−SUM OR Y−SUM ?_
Enter Y ↵ for Y subtotals.	12	Y ↵ ↵ ↵ ↵	Y=1 SUM=1047 Y=2 SUM=429 Y=3 SUM=2283 GRAND T. =3759
The menu display	13	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_

	Step	Key operation	Display
(3) Sorting If the printer is connected, enter Y ↵.	14	3 ↵	PRINTER ON ? (Y/N)_
Arrangement for each subtotal (X or Y)	15	N ↵	SORT X OR Y ?_
Sorting is being executed.	16	X ↵	SORTING NOW
The data sorted in descending order is output, together with ranking, item name, subtotal and percentage.	17	↵ ↵ ↵ ↵	1 X=2 1378 37% 2 X=1 1322 35% 3 X=3 1059 28% GRAND T. 3759
The menu display	18	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
If the printer is connected, enter Y ↵.	19	3 ↵	PRINTER ON ? (Y/N)_
When sorting data in item Y, enter Y ↵.	20	N ↵	SORT X OR Y
	21	Y ↵	SORTING NOW
The data sorted in descending order is output.	22	↵ ↵ ↵ ↵	1 Y=3 2283 61% 2 Y=1 1047 28% 3 Y=2 429 11% GRAND T. 3759
	23	↵	1 DATA INPUT 2 TOTAL 3 SORT 4 DATA OUTPUT ?_
(4) Data output If the printer is connected, enter Y ↵	24	4 ↵	PRINTER ON ? (Y/N)_
	25	N ↵ ↵ ↵ ↵	X=1 Y=1 DATA=321 X=1 Y=1 DATA=369 X=1 Y=1 DATA=357 X=1 Y=2 DATA=159 ⋮

* The sum of percentages is not always 100% depending on data values.

VARIOUS GRAPH MAKING

This program draws various types of graphs with the plotter-printer (FA-10). Up to 12 data items can be input. The range of data is as follows:

| Value of data | ≦ 1E90

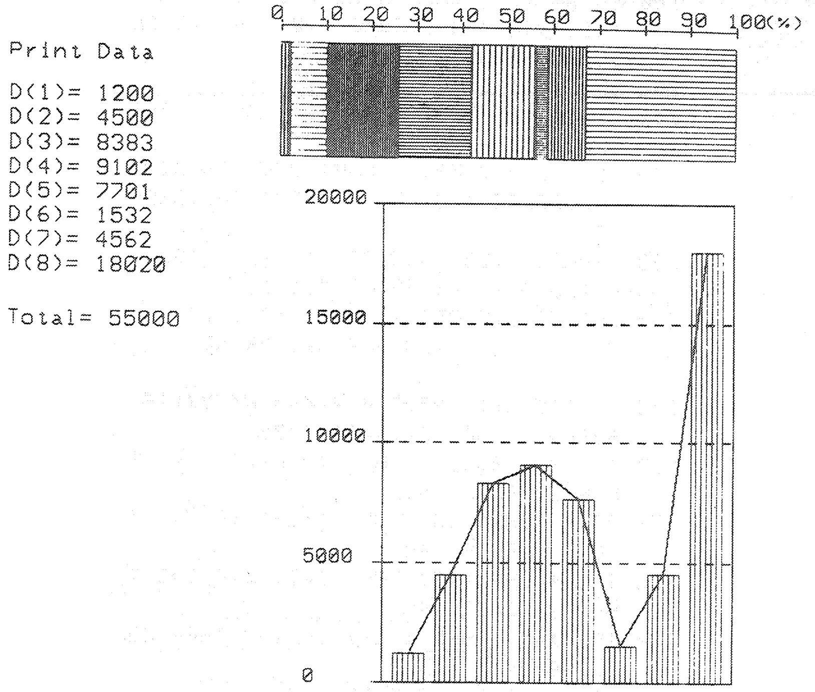
The program can draw beautiful band, bar, and line graphs, taking advantage of the 4-color plotter-printer.

* This program is stored on the microcassette tape which comes with the optional microcassette tape recorder (CM-1).

Explanation

When the program is executed, the menu is displayed. First, data must be entered. This can be done by pressing 1.
The range of data is shown above. Data may be negative numerical data. Up to 12 data items can be entered. After the 12th data item is entered, the menu display automatically appears. To terminate the input in the middle, press the ↵ key without entering any numerical data.
Menu 2 is used to correct input data. Pressing 2 causes the first input data to be displayed, and pressing the ↵ key causes the next data to appear. Pressing the SHIFT and ↵ keys causes the previous data to appear. Input the correct data when data to be corrected appears.
Menu 3 is the routine to make graphs. Three kinds of graph names are displayed by pressing 3. Select the type of graph by pressing 1, 2, or 3. (Pressing 4 causes return to the menu display.)
Type 1 is band graph. The entire length of band represents 100%, with the percentage of each data element represented by the length it occupies. For easy recognition, the individual data elements are shown in different colors and stripes. When the band graph is selected, negative data causes the menu to be displayed.
Type 2 is bar graph. The scale is automatically set according to the size of input data. In this graph, positive values are output in green and negative values are output in red.
Type 3 is line graph. When 3 is pressed, "Overwrite on the Bar Graph?" is displayed. If a bar graph has been drawn just before, a line graph can be overwritten on the bar graph by pressing Y. If no bar graph has been drawn or a line graph should not be overwritten on a bar graph, press N. In this case, the appropriate scale is automatically set and the line graph is drawn.
Regardless of the type selected, the menu display appears after the graph is drawn. The menu display also appears when no data is entered. Note that starting the program again clears the existing data.
Menu 4 terminates program execution.
Menu 5 is provided to output the data to the plotter-printer. It can also be used to output the total of data.

Print-out Example



```
Print Data

D(1)= 1200
D(2)= 4500
D(3)= 8383
D(4)= 9102
D(5)= 7701
D(6)= 1532
D(7)= 4562
D(8)= 18020

Total= 55000
```

Program

```
 10 CLEAR
 20 CLS :PRINT "    ---- DATA ----"
 30 PRINT TAB(2);"1:Input 2:Correct",T
    AB(2);"3:Graph 4:END",TAB(2);"5:Pr
    int Data";
 40 K=VAL(INKEY$):IF K<1 THEN 40 ELSE
    IF K>5 THEN 40
 50 BEEP :GOTO K*100
100 CLS :ERASE D:DIM D(13):Z=1
110 LOCATE 2,3:PRINT "<RETURN> : END";
```

```
120 LOCATE 6,0:PRINT "                    ";
130 LOCATE 0,0:PRINT "D(";Z;")=";:INPU
    T "",AB$
140 IF AB$<>"" THEN D(Z)=VAL(AB$):IF Z
    <12 THEN Z=Z+1:GOTO 120 ELSE 20 EL
    SE Z=Z-1:GOTO 20
200 IF Z<1 THEN 20
210 CLS :I=1:PRINT TAB(46);":Shift RET
    URN",TAB(6);":RETURN ";
220 LOCATE 4,2:PRINT CHR$(228);CHR$(22
    9):LOCATE 4,3:PRINT CHR$(230);CHR$
    (231);
230 DRAWC(32,23)-(47,23)-(47,24)-(32,2
    4):LOCATE 0,0:PRINT "D(";
240 LOCATE 2,0:PRINT I;")=";D(I);"
                 ":LOCATE 6,1:PRINT "
                 "

250 LOCATE 6,1:INPUT AB$:K$=INKEY$:IF
    AB$<>"" THEN D(I)=VAL(AB$)
260 IF K$=CHR$(24) THEN I=I-1:IF I<1 T
    HEN 20 ELSE 240
270 IF K$=CHR$(13) THEN I=I+1:IF I>Z T
    HEN 20 ELSE 240
280 IF K$=CHR$(23) THEN I=I+1:IF I>Z T
    HEN 20 ELSE 240
290 IF K$="" THEN I=I+1:IF I>Z THEN 20
     ELSE 240
300 IF Z<1 THEN 20 ELSE CLS
310 PRINT TAB(5);"1:Band",TAB(5);"2:Ba
    r"
320 PRINT TAB(5);"3:Line",TAB(5);"4:ME
    NU";
330 K=VAL(INKEY$):IF K<1 THEN 330 ELSE
     IF K>4 THEN 330
340 BEEP :GOTO K*1000
400 LPRINT CHR$(28);CHR$(46):END
500 IF Z<1 THEN 20 ELSE GOSUB 6000:LPR
    INT "Q1":V=0
510 LPRINT "M93,0","PPrint Data"
520 FOR I=1 TO Z
530 LPRINT "M";90-4*I;",0","PD(";MID$(
    STR$(I),2);")=";D(I)
540 V=V+D(I):NEXT I
```

```
 550 LPRINT "M";90-I*4-5;",0","PTotal="
     :V:GOSUB 6000:GOTO 20
1000 GOSUB 6000:A=0
1010 FOR I=1 TO Z:IF D(I)<0 THEN ERASE
     I:GOTO 20 ELSE A=A+D(I):NEXT I
1020 IF A<=0 THEN 20
1030 LPRINT "Q5,0","X1,8,10","M85,1","P
     (";CHR$(37);")"
1040 FOR I=100 TO 0 STEP -10:LPRINT "M"
     ;-4+8*I/10;",1","P";I:NEXT I
1050 B=0:C=0
1060 FOR I=1 TO Z
1070 C=ROUND(D(I)/A*80+C,-2)
1080 LPRINT "J";I MOD 4,"A";B;",-3,";C;
     ",-23"
1090 LPRINT "G";I MOD 2+1;",";C-B;",-20
     ,";(I MOD 3)/4+.5
1100 B=C:NEXT I
1110 LPRINT "H30":GOTO 20
2000 GOSUB 6000:GOSUB 7000:GOSUB 8000
2010 FOR I=1 TO Z
2020 IF D(I)>=0 THEN J=2 ELSE J=3
2030 LPRINT "J";J
2040 LPRINT "A";O;",";6-8*I;",";ROUND(D
     (I)/A*90/N+O,-2);",";-8*I
2050 LPRINT "M";O;",";6-8*I,"G1,";ROUND
     (D(I)/A*90/N,-2);",-6"
2060 NEXT I
2070 LPRINT "M0,";-8*(Z+1):GOTO 20
3000 IF Z<2 THEN 20 ELSE CLS :PRINT "Ov
erwrite on the Bar Graph?";TAB(8);"Y/N"
3010 K$=INKEY$:IF K$="Y" THEN BEEP :GOT
     O 3080 ELSE IF K$<>"N" THEN 3010 E
     LSE BEEP
3020 GOSUB 6000
3030 GOSUB 7000:GOSUB 8000
3040 LPRINT "L1","J0"
3050 FOR I=1 TO Z
3060 F=3-8*I:IF I MOD 2=1 THEN LPRINT "
     D0,";F;",90,";F ELSE LPRINT "D90,"
     ;F;",0,";F
3070 NEXT I:LPRINT "L0","M0,0"
3080 S=S+1:IF S>3 THEN S=0
3090 T=T+.25:IF T>3.9 THEN T=0
```

```
3100 IF T<2 THEN LPRINT "B1.6" ELSE LPR
     INT "B6.4"
3110 LPRINT "J";S,"L";T
3120 G=ROUND(D(1)/A*90/N+O,-2):H=-5
3130 FOR I=2 TO Z
3140 U=ROUND(D(I)/A*90/N+O,-2):V=3-8*I
3150 LPRINT "D";G;",";H;",";U;",";V
3160 G=U:H=V
3170 NEXT I
3180 LPRINT "B3.2","M0,";-8*(Z+1):GOTO
     20
4000 GOTO 20
6000 LPRINT CHR$(28);CHR$(37),"O0,0","J
     0","L0","S1","Q0","Y0","B3.2","H20
     "
6010 S=0:T=0:RETURN
7000 Y=-9E99:B=9E99
7010 FOR I=1 TO Z
7020 IF D(I)>Y THEN Y=D(I)
7030 IF D(I)<B THEN B=D(I)
7040 NEXT I
7050 IF Y>=0 THEN D(0)=Y ELSE D(0)=0
7060 IF B>0 THEN D(Z+1)=0 ELSE D(Z+1)=B
7070 RETURN
8000 IF SGND(0)*SGND(Z+1)<=0 THEN M=ABS
     (D(0)-D(Z+1)) ELSE M=D(0):IF M<0 T
     HEN M=ABSD(Z+1)
8010 IF M=<0 THEN 20 ELSE R=INTLGTM:A=1
     0^R
8020 IF A*INT(M/A)*.75<M THEN A=A*.5
8030 D=LEN(STR$(A))*2.4+5
8040 IF SGND(0)*SGND(Z+1)<0 THEN N=INT(
     M/A)+2 ELSE N=INT(M/A)+1
8050 C=ABSINT(D(0)/A)+SGND(0)
8060 FOR I=N TO 0 STEP -1:IF C=0 THEN O
     =I*90/N
8070 C=C-1:NEXT I
8080 LPRINT "D0,0,90,0","Q1":W=18/N:V=0
8090 V=V+5:IF V*W<18 THEN 8090
8100 IF D(0)<=0 THEN 8150 ELSE X=O
8110 K=ROUND(X,-2):LPRINT "D";K;",2,";K
     ;",-2"
```

```
8120 LPRINT "M";K;",";D,"P";ROUND((X-O)
     *A*N/90,R-2)
8130 LPRINT "L1","D";K;",0,";K;",";-Z*8
     -2,"L0":X=X+W*V
8140 IF X<90 THEN 8110
8150 LPRINT "D90,2,90,-2","M90,";D,"P";
     ROUND((90-O)*A*N/90,R-2)
8160 IF D(Z+1)>=0 THEN 8220 ELSE X=O-W*
     V
8170 K=ROUND(X,-2):LPRINT "D";K;",2,";K
     ;",-2"
8180 LPRINT "M";K;",";D,"P";ROUND((X-O)
     *A*N/90,R-2)
8190 LPRINT "L1","D";K;",0,";K;",";-Z*8
     -2,"L0":X=X-W*V
8200 IF X>0 THEN 8170
8210 LPRINT "D0,2,0,-2","M0,";D,"P";ROU
     ND(-O*A*N/90,R-2)
8220 LPRINT "D0,0,0,";-8*Z-2;",90,";-8*
     Z-2;",90,0","M0,0"
8230 RETURN
```

In regard to inputting line 30, 140, 240, 3010, 3060 and 8000, if one-key commands are used, spaces are automatically made after the commands and a whole line cannot be input because the input range of 79 characters is exceeded. Therefore, input each line after deleting extra spaces by using the DEL key.

Example: 30␣PRINT␣TAB(2);. . . .

⇩

30 PRINTTAB(2);. . . .

Operation

	Step	Key operation	Display
Menu display		SHIFT P0	---- DATA ---- 1 : Input 2 : Correct 3 : Graph 4 : END 5 : Print Data
(1) Data input	1	1	D(1) = _ 〈RETURN〉 : END
	2	2112 ⏎ ⋮	D(2) = _ ⋮
After the 12th data item is entered, the menu display automatically appears. (2) Data correction After the correction is made to the last, the menu is displayed.	3	2 1200 ⏎ ⋮	D(1) = 1200 ? _ ▲Shift RETURN (Previous data) ▼RETURN (Next data)
(3) Graph making	4	3	1 : Band 2 : Bar 3 : Line 4 : MENU
Band graph is printed out. Bar graph is printed out. Determine whether a line graph is over written on the bar graph or not. Line graph is printed out.	5	1 2 3 Y or N	(After the graph is output, the menu is displayed.) (After the graph is output, the menu is displayed.) Overwrite on the Bar Graph ? Y/N (After the graph is output, the menu is displayed.)
(5) Data output. Data are printed out.	6	5	(After the data are output, the menu is displayed.)
(4) Execution termination.	7	4	

WORLD STANDARD TIME

This program indicates the difference between the Greenwich standard time and the time in a specified place on the globe.

Explanation

This program applies the principle of character string. It stores the combination of DATA and READ statements in a character variable by the FOR ~ NEXT statement.
There are four types of character variables: fixed character variable, registered character variable, fixed-length character array variable, and defined-length character array variable. The program uses the first three types of character variables.

When the program is executed, it first outputs a message, then sequentially outputs the time differences and place names stored in a character array. Press the ⏎ key when "NEXT" is displayed on the screen. After all the data has been output, you are requested to enter a place name. Enter the place name, and the program displays "ABOVE OK (Y/N)? " If the input data is correct, press the Y key. If you wish to correct the input data, press the N key. If the place name entered has not been stored, "NOT FOUND" is displayed and the program requests to re-enter another place name. When the place name entered is correct, the time difference between Greenwich and that place is displayed, and a request for input is made again.
Note that this program simply displays data as it is. By using the appropriate functions, such as VAL and STR$, it will be possible, for example, to enter the Japan time to cause the time in another place to be displayed. Just try.
Also note that the program has an endless loop formed by the GOTO statement. Therefore press the BRK key when terminating the program execution.

Program

```
  10 CLS
  20 PRINT "WORLD STANDARD TIME"
  30 BEEP
  40 DIM A$(15),B$(15)
  50 FOR I=0 TO 15
  60 READ A$(I)
  70 NEXT I
  80 DATA"0","-12","-9","-8","-7","-5",
"-4","-3","-2","-1","0"
  90 DATA"+2","+3","+4","+5","+6"
 100 FOR I=0 TO 15
 110 READ B$(I)
 120 NEXT I
 130 DATA"WORLD","NEW ZEALAND","JAPAN",
"HONG KONG","SUMATRA"
 140 DATA"PAKISTAN","SAUDI ARABIA","ETH
IOPIA","ISRAEL","EUROPE"
 150 DATA"GREAT BRITAIN","GREENLAND","A
RGENTINA","VENEZUELA","COLOMBIA","MEXICO
"
 190 PRINT "DATA":BEEP
 200 FOR I=0 TO 15
 210 PRINT A$(I);"h ";B$(I)
 220 INPUT "NEXT",C$
 230 NEXT I
 240 CLS
 250 INPUT "NAME";CA$
 260 INPUT "ABOVE OK(Y/N)";D$
 270 IF D$="N" THEN 240
 280 FOR I=0 TO 15
 290 IF CA$=B$(I) THEN J=I:GOTO 320
 300 NEXT I
 310 PRINT "NOT FOUND":BEEP :GOTO 240
 320 BEEP 0:BEEP 1
 330 PRINT A$(J);"h ";B$(J)
 340 GOTO 250
 350 END
```

Operation

	Step	Key operation	Display
Indicates the Greenwich standard time. →		SHIFT PO 0	WORLD STANDARD TIME DATA 0h WORLD NEXT _
	1	↵	−12h NEW ZEALAND NEXT _
	2 ⋮ 15	↵ ⋮ ↵	−9h JAPAN NEXT _ ⋮ +6h MEXICO NEXT _
Input of a place name →	16	↵	NAME ? _
Input of a place name which has been stored. →	17	JAPAN ↵	ABOVE OK (Y/N) ? _
Entering N causes return to step 16. → Requests a place name again. →	18	Y ↵	−9h JAPAN NAME ? _

Variable contents	
A$	Data of time difference
B$	Data of place name
C$	For displaying next data
CA$	Place name to be input
D$	Y or N
I	Control variable
J	Subscript

CHAPTER 6

REFERENCE MATERIAL

6-1 PB-700 COMMAND TABLE

6-1-1 Operational Symbols

1 Arithmetic operators

Name	General format	PB-700 format	Meaning	Priority
Power	x^y	X ^ Y	Raise X to power Y.	1
Multiplication	$x \times y$	X *Y	Multiply X by Y.	2
Division	$x \div y$	X / Y	Divide X by Y.	2
Remainder	$x \div y = z$	X MOD Y	What is left undivided when X is divided by Y.	3
Addition	$x + y$	X + Y	Add Y to X.	4
Subtraction	$x - y$	X − Y	Subtract Y from X.	4
Assignment	$x = y + 5$	X = Y + 5	Assign Y + 5 to X.	5

2 Relational operators (conditional expressions)

General format	PB-700 format	Meaning
$x = y$	X = Y	X is equal to Y.
$x \neq y$	X<>Y, X><Y	X is not equal to Y.
$x < y$	X < Y	X is smaller than Y.
$x > y$	X > Y	X is greater than Y.
$x \leqq y$	X<=Y, X=<Y	X is smaller than, or equal to, Y.
$x \geqq y$	X>=Y, X=>Y	X is greater than, or equal to, Y.

- The relational operators are valid only in IF statements.
- A comparison can be made between numerical constants, numerical variables, and numerical expressions, while a comparison can be made between character constants and character variables.

3 Character expression operators

+ ····· Two or more character strings can be concatenated by a + (plus sign).

6-1-2 Special Character

General format	PB-700 format	Meaning
$x \times 10^y$	X E Y	Exponent entry (Multiply number X by Y power of 10.)

- If the result of operation is equal to, or greater than, 10^{10} or smaller than 10^{-3}, it is automatically indicated by exponential notation.

6-1-3 Built-in Functions

Name	General format	PB-700 format	Meaning/remarks	Page
Trigonometric functions	$\sin x$	SIN X	Gives the sine of X (−5400 < X < 5400)	222
	$\cos x$	COS X	Gives the cosine of X (−5400 < X < 5400)	225
	$\tan x$	TAN X	Gives the tangent of X (−5400 < X < 5400)	226
Inverse trigonometric functions	$\sin^{-1} x$ $\cos^{-1} x$ $\tan^{-1} x$	ASN X ACS X ATN X	Given the arcsine of X (|X| ≦ 1) Gives the arccosine of X (|X| ≦ 1) Gives the arctangent of X (|X| < 10^{100})	227
Logarithmic functions	$\log x$ $\ln x$	LOG X LGT X	$\log_{10}$ X (common logarithm) log e X (natural logarithm)	230
Exponential functions	e^x x^y	EXP X X ^ Y	X power of natural logarithm base (e) Y power of X.	233
Square root	$\sqrt{x}$	SQR X	Gives the square root (X ≧ 0)	229
Absolute value	$\lvert x \rvert$	ABS X	Gives the absolute value of X.	235
Integer		INT X	When X>0, the fraction portion of X is discarded. When X<0, the fraction portion of |X| is rounded up, then a − (minus sign) is prefixed to |X|. INT 1.2→1 INT −1.2→−2	237
Fraction		FRAC X	Eliminates the integer portion to obtain only the fraction portion. The fraction portion is prefixed with either a + (plus sign) or a − (minus sign), whichever is appropriate.	239
Circular constant	π	PI	Gives an approximate ratio of the circumference of a circle to its diameter in 11 digits, that is, 3.1415926536.	245
Random number		RND	Generates a 10-digit pseudo random number (0 < RND < 1).	246
Sign		SGN X	Checks the sign of an argument: X < 0→ −1 X = 0→ 0 X > 0→ 1	241
Rounding		ROUND (X, Y)	Rounds up or off the value of X to 10^{Y} positions (Y < 100).	243

6-1-4 Character Functions

Use	Command	Example	Meaning	Page
Gives decimal code of first one character.	ASC	PRINT ASC ("E")	Displays the decimal code of character E.	248
Gives one character in the character code.	CHR $	PRINT CHR $ (69)	Displays character E of character code 69.	250
Converts numeral in a character string to numerical value.	VAL	A = VAL (X$)	Converts a character string of numerals stored in character variable X$ to a numerical value.	252
Converts numerical value to character string.	STR $	C$ = STR $ (X)	Converts a numerical value stored in numerical variable X to a character string of numerals.	255
Fetches specified number of characters from left of character string.	LEFT $	C$ = LEFT $ (X$, 3)	Fetches three characters from the left of a character string stored in X$ and assigns them to C$.	257
Fetches specified number of characters from right of character string.	RIGHT $	C$ = RIGHT $ (X$, 3)	Fetches three characters from the right of a character string stored in X$ and assigns them to C$.	258
Fetches specified number of characters.	MID $	C$ = MID $ (X$, 3, 5)	Fetches five characters starting from the third character of a character string stored in X$ and assigns them to C$.	259
Counts the number of characters in a character string.	LEN	A = LEN (X$)	Assigns the number of characters in a character string stored in X$ to A.	261
Inputs one character from the keyboard.	INKEY $	A$ = INKEY $	When INKEY$ is executed, if one key on the keyboard has been pressed, it is assigned to A$. Only one character can be assigned to A$.	262

6-1-5 Other Functions

Use	Command	Example	Meaning	Page
Checks whether a dot on the screen is on or off.	POINT	POINT (10, 20)	Checks whether the dot represented by coordinates (10, 20) on the screen is on or off, and displays '1' if the dot is on and displays '0' if the dot is off.	271
Moves cursor by specified number of positions.	TAB	PRINT TAB(10)	Moves the cursor to the 10th position on the screen.	265
Specifies output format.	USING	PRINT USING "###.##"; A	Displays a numerical value stored in numerical variable A in the format "###.##".	267

6-1-6 Manual Commands

Use	Command	Example	Meaning	Page
Resumes program execution.	CONT	CONT	Resumes the execution of a program that has been stopped by a STOP statement or by the STOP ANS key.	136
Deletes program.	DELETE	DELETE 50	Deletes line 50 of a program.	138
		DELETE 30–	Deletes the lines of a program from line 30 to the last.	
		DELETE –100	Deletes lines of a program up to line 100 from the beginning.	
		DELETE 150 – 200	Deletes lines of a program from line 150 to line 200.	
Modifies program.	EDIT	EDIT	Displays the first line and specifies the EDIT mode.	141
		EDIT 30	Displays line 30 and specifies the EDIT mode.	
Lists program.	LIST	LIST	Displays the program in the presently specified program area.	145
		LIST 50	Displays line 50 of a program.	
		LIST 30 –	Displays the lines of a program from line 30 to the last.	
		LIST – 50	Displays lines up to line 50.	
		LIST 30 – 50	Displays lines from line 30 to line 50.	
		LIST ALL	Displays programs in the entire program area.	
		LIST V	Displays registered variable names.	

Use	Command	Example	Meaning	Page
Prints program.	LLIST	LLIST	Prints a program in the presently specified program area.	145
		LLIST 50	Prints line 50.	
		LLIST 30 –	Prints lines from line 30 to the last.	
		LLIST – 50	Prints lines from the beginning to line 50.	
		LLIST 30 – 50	Prints lines from line 30 to line 50.	
		LLIST ALL	Prints programs in all program areas.	
		LLIST V	Prints registered variable names.	
Reads program from cassette tape to the PB-700.	LOAD	LOAD	Reads a program with internal code format to the presently specified program area.	149
		LOAD "file name"	Reads a program which has the specified file name with internal code format to the presently specified program area.	
		LOAD ALL	Reads programs with internal code format to the entire program areas.	
		LOAD ALL "file name"	Reads programs which have the specified file name with internal code format to the entire program areas.	
		LOAD, A	Reads a program with ASCII code format to the presently specified program area.	
		LOAD "file name", A	Reads a program which has the specified file name with ASCII code format to the presently specified program area.	
		LOAD, M	Mixes the program in the presently specified program area with the program read with ASCII code format.	
		LOAD "file name", M	Mixes the program in the presently specified program area with the program which has the specified file name read with ASCII code format.	
Erases program.	NEW	NEW	Erases a program in the presently specified program area.	153
		NEW ALL	Clears the entire RAM for initialization.	

Use	Command	Example	Meaning	Page
Protects program.	PASS	PASS "KEY"	Sets a password named "KEY."	155
Specifies a program area.	PROG	PROG 2	Specifies the program area 2.	157
Starts program execution.	RUN	RUN	Starts the execution of a program from the beginning of the presently specified program area.	158
		RUN 100	Starts the execution of a program from line 100.	
Stores programs to a cassette tape.	SAVE	SAVE	Stores the program in the presently specified program area to a cassette tape with the internal code format.	159
		SAVE ALL	Stores all programs in all program areas to a cassette tape with the internal code format.	
		SAVE, A	Stores the program in the presently specified program area to a cassette tape with the ASCII code format.	
		SAVE "ABC" SAVE ALL "CASIO" SAVE "TEST", A	Stores a program with its file name to a cassette tape.	
Displays status of program areas.	SYSTEM	SYSTEM	Displays the status of the program areas, the number of remaining bytes, and the angle unit (ANGLE).	162
Checks programs stored on a cassette tape.	VERIFY	VERIFY	Performs a parity check of a program file which appears first.	163
		VERIFY "ABC"	Performs a parity check of the program with a specified file name.	

6-1-7 Program Commands

Use	Command	Example	Meaning	Page
Specifies angle unit.	ANGLE	ANGLE 0	Specifies degree as the angle unit.	164
		ANGLE 1	Specifies radian as the angle unit.	
		ANGLE 2	Specifies gradient as the angle unit.	
Generates buzzer sound.	BEEP	BEEP	Same as BEEP 0.	165
		BEEP 0	Generates a buzzer sound ('boo') once.	
		BEEP 1	Generates a buzzer sound ('pee') once.	

Use	Command	Example	Meaning	Page
Reads and executes program.	CHAIN	CHAIN	Loads and executes PF B that first appears.	166
		CHAIN "XYZ"	Loads and executes the program with a specified file name.	
Clears variables.	CLEAR	CLEAR	Clears all variables.	168
Clears display screen	CLS	CLS	Clears the entire screen and moves the cursor to the home position.	169
Stores data.	DATA	DATA 1,2,3	Stores data to be referenced by the READ statement.	212
Declares array.	DIM	DIM A (3)	Declares a one-dimensional single-precision numerical array.	170
		DIM B (2, 3)	Declares a two-dimensional single-precision numerical array.	
		DIM C! (4)	Declares a one-dimensional half-precision numerical array.	
		DIM D! (3, 4)	Declares a two-dimensional half-precision numerical array.	
		DIM E$ (5)	Declares a one-dimensional fixed-length character array.	
		DIM F$ (4,5)	Declares a two-dimensional fixed-length character array.	
		DIM G$ (2)*3	Declares a one-dimensional defined-length character array and specifies 3 as the character length.	
		DIM H$ (4,5)*6	Declares a two-dimensional defined-length character array and specifies 6 as the character length.	
Draws point and straight line.	DRAW	DRAW (0,0)	Draws a point at coordinates (0, 0).	175
		DRAW (1,0)−(5,10)	Draws a straight line from coordinates (1, 0) to (5, 10).	
Erases point and straight line.	DRAWC	DRAWC (0,0)	Erases the point at coordinates (0, 0).	175
		DRAWC (1,0)−(5,10)	Erases the straight line between coordinates (1, 0) and (5, 10).	
Terminates program execution.	END	END	Terminates the execution of a program.	178
Deletes array name.	ERASE	ERASE A	Deletes the registered array variable A.	179
Loop (repeat).	FOR TO STEP ≀ NEXT	FOR I=5 TO 20 STEP 0.5 ≀ NEXT I	Repeatedly performs the processing between FOR and NEXT while incrementing the value of variable I by 0.5 from 5 up to 20.	180

Use	Command	Example	Meaning	Page
Reads variable data from cassette tape.	GET	GET A	Reads the variable data that first appears.	185
		GET "MAX" B	Reads the variable data having a file name "MAX."	
Jumps to subroutine.	GOSUB	GOSUB 100	Jumps to the subroutine on line 100.	188
		GOSUB PROG 3	Jumps to the subroutine in program area 3.	
End of subroutine.	RETURN	RETURN	Returns to the command following the GOSUB statement.	188
Unconditional jump.	GOTO	GOTO 500	Jumps to line 500.	192
		GOTO PROG 5	Jumps to program area 5.	
Conditional jump.	IF THEN ~ ELSE ~	IF I>9 THEN 50 ELSE 80	Jumps to line 50 if I is greater than 9; otherwise, jumps to line 80.	194
Data input from keyboard.	INPUT	INPUT S	Displays a ?, then waits for data to be entered in S.	197
		INPUT "NAME", T$	Displays NAME, then waits for data to be entered in T$.	
		INPUT "NAME"; U$	Displays NAME ?, then waits for data to be entered in U$.	
Assigns data to variable.	LET	LET A=B	Assigns B to A.	203
Specifies cursor position.	LOCATE	LOCATE 2, 3	Specifies coordinates (2, 3) as the cursor position.	204
Displays data.	PRINT	PRINT C, D	Displays the values of C and D in alternate lines.	205
		PRINT C; D	Displays the values of C and D in succession.	
Prints data.	LPRINT	LPRINT C, D	Prints the values of C and D in alternate lines.	205
		LPRINT C; D	Prints the values of C and D in succession.	
Stores variable data to cassette tape.	PUT	PUT A	Stores data of variable A to cassette tape.	210
		PUT "DATA" A	Stores data of variable A with file name to cassette tape.	
Reads stored data.	READ	READ X	Reads data stored in variable X by DATA statement.	212
Comment.	REM	REM ***	Writes a comment to a program.	217

Use	Command	Example	Meaning	Page
Changes sequence of execution of DATA statements.	RESTORE	RESTORE	Reads from the first DATA statement when executing the next READ statement.	212
		RESTORE 100	Reads from the DATA statement on line 100 when executing the next READ statement.	
Halts program execution.	STOP	STOP	Halts the execution of a program.	218
Traces program execution.	TRON	TRON	Traces the status of program execution.	220
Releases tracing of program-execution.	TROFF	TROFF	Releases TRON mode.	220

6-2 ERROR MESSAGE TABLE

Error message	Content of error	Corrective measure
BS error (Bad Subscript)	● The subscript of an array variable is a negative value or the value exceeds 255. **Example)** DIM A (256) ● The specified numerical value is outside the argument range. **Example)** The POINT function has the following argument ranges: $0 \leqq X \leqq 159$ and $0 \leqq Y \leqq 31$ But the specified numerical value is outside this range.	● Change the value of subscript to a value within the specified range. When the subscript is a variable, check the associated part. ● Re-specify the subscript within the specified argument range.
BV error (Buffer oVerflow)	● An overflow of the input and output buffer occurred.	● Each operation or program statement must not exceed 79 characters in length.
DA error (DAta error)	● A READ statement and a GET statement were executed even if there is no data to read.	● Check the relation between the READ and DATA statements. Create data in the statement to be assigned to the variable.
DD error (Duplicate Definition)	● Arrays having the same array name and a different subscript were doubly defined. **Example)** The following array variable declarations by a DIM statement cause a DD error. .... DIM A(1), A(2, 3) DIM A! (1), A!(2, 3) DIM A$(1), A$(2, 3) DIM A$(1)*20, A$(2, 3)*20 DIM A$(1)*20, A$(2, 3)*20	● Check the variable on the line in which an error has occurred. Also check subscripts of the same array name. If there are different subscripts, change either of the array names and reorganize the program.
FC error (illegal Function Call)	● An attempt was made to execute any of the following manual commands in a program: CONT, PASS, RUN, EDIT, DELETE ● An attempt was made to execute any of the following program commands manually: END, LET, REM, STOP, LOCATE, DRAW, DRAWC, GOTO, GOSUB, RETURN, INPUT, DATA, READ, RESTORE, FOR~NEXT, IF~THEN~ELSE~ ● A CONT command was used when program execution could not be resumed.	● Remove the incorrect command from the program, then reorganize the program. ● Executes the command with a line number attached in a program. ● Press the BRK key, then either re-execute the program from the beginning or, if the stopped line is known, re-execute the program from the line next to the stopped line by "RUN The line number".

Error message	Content of error	Corrective measure
FO error (NEXT without FOr)	● There is no FOR statement corresponding to a NEXT statement.	● Check the nesting structure.
GS error (RETURN without GoSub)	● There is a RETURN statement not corresponding to a GOSUB statement.	● Check the nesting structure in the GOSUB statement, and clearly distinguish between the main routine and the subroutine.
MA error (MAthematical error)	● An arithmetic operation involving numerical values or numerical functions is uncertain or impossible. **Example)** Division by 0.	● Check the numerical expression on the line in which an error has occurred. Also check the value of the associated variable.
NO error (Nesting Overflow)	● The number of nesting levels exceeded the specified limit. **Example)** GOSUB~RETURN Max. 12 levels FOR~NEXT Max. 6 levels	● Check the nesting structure and reduce the nesting level within the allowable range.
NR error (device Not Ready)	● The I/O device is not correctly connected. **Example)** No magnetic tape recorder is connected.	● Check that the relevant device is properly connected and switched on.
OM error (Out of Memory)	● The RAM capacity is insufficient. **Example)** * If the RAM capacity becomes insufficient during a load operation, an OM error occurs with the program that is already loaded remaining in RAM. * An OM error occurs if an array declaration is made when the RAM capacity is insufficient.	● Clear unnecessary programs. Increase the memory capacity by RAM expansion pack. Check the number of remaining bytes by the SYSTEM command.
OV error (OVerflow error)	● The result of an operation or the numerical value entered exceeds 10^{100}.	● Check the numerical expression on the line in which an error has occurred. Insert a PRINT statement in the program to check the value of the variable.
PR error (PRotected error)	● An attempt was made to execute any of the following commands which cannot be used with a program having a password: DELETE, LIST, LLIST, NEW, EDIT ● An attempt was made to add a new line to, or delete a line from, a program having a password.	● Re-enter the password, release the locked condition, and execute the program.
	● A different password was entered.	● Input the correct password.
	● An attempt was made to load a program whose password is different from the PB-700 password.	● Release the PB-700 password before loading. In this case, the newly loaded password becomes the PB-700 password.

Error message	Content of error	Corrective measure
RW error (Read Write error)	● A parity error occurred during execution of a LOAD or VERIFY command.	● Store the program again by the SAVE command.
SN error (SyNtax error)	● The command format has an error.	● Use the EDIT command to call the line in which an error has occurred and correct the line.
	● The line number has a fraction.	● Correct the line number.
	● An array having three or more dimensions was declared.	● The number of dimensions must not exceed 2.
SO error (Stack Over error)	● The stack of numerical value exceeded 8 levels. ● The operator stack exceeded 20 levels.	● Simplify or divide the numerical expression so that the stack level can fall within the specified range.
	● The character stack exceeded 10 levels.	● Simplify or divide the character expression so that the stack level can fall within the specified range.
ST error (STring error)	● An attempt was made to assign a character string whose length exceeds the allowable character variable length, to the character variable. The allowable character variable length is as follows: – Fixed character variable Max. 7 characters – Registered character variable Max. 16 characters – Fixed-length character array variable Max. 16 characters – Defined-length character array variable Max. 79 characters	● Change the variable to another variable that can contain more characters. Decrease the number of characters of the character string to be assigned to the variable. Be careful when concatenating character strings.
TM error (Type Missmatch)	● In an assignment statement, the left and right sides have a different variable type.	● Both the left and right sides of an assignment statement must be either numerical variables or character variables.
	● The argument types do not match during an assignment.	● Assign a numerical value to a numerical variable, and a character string to a character variable.
UL error (Undefined Line number)	● The specified line number does not exist in an IF~THEN, GOTO or GOSUB statement.	● Create a line number to which a jump is to be made or change the specified line number to which a jump is to be made.
	● No program exists in the program area specified by a GOTO or GOSUB statement.	● Create a program area to which a jump is to be made, or change the location to which a jump is to be made.

Error message	Content of error	Corrective measure
UV error (Undefined Variable)	● An undefined variable was used.	● Check the initial value of a variable used.
	● An array variable was used without declaring it by the DIM statement.	● Declare the array by the DIM statement at the beginning of the program.
	● A subscript of an array variable exceeds the range specified by DIM.	● Change the size of the subscript within the specified range.
VA error (VAriable error)	● An attempt was made to register more than 40 variables.	● Up to 40 registered and array variables can be used. Check the variable names by LISTV, and delete unnecessary variable names by CLEAR or ERASE.

6-3 CHARACTER CODE TABLE

0		32	(SPC)	64	@	96	`	128	▁	160	(SPC)	192	タ	224	═
1		33	!	65	A	97	a	129	▂	161	。	193	チ	225	╞
2		34	"	66	B	98	b	130	▃	162	「	194	ツ	226	╪
3		35	#	67	C	99	c	131	▄	163	」	195	テ	227	╡
4		36	$	68	D	100	d	132	▅	164	、	196	ト	228	◢
5		37	%	69	E	101	e	133	▆	165	・	197	ナ	229	◣
6		38	&	70	F	102	f	134	▇	166	ヲ	198	ニ	230	◥
7		39	'	71	G	103	g	135	█	167	ァ	199	ヌ	231	◤
8		40	(	72	H	104	h	136	▏	168	ィ	200	ネ	232	♠
9		41	)	73	I	105	i	137	▎	169	ゥ	201	ノ	233	♥
10		42	*	74	J	106	j	138	▍	170	ェ	202	ハ	234	♦
11	*(HOME)	43	+	75	K	107	k	139	▌	171	ォ	203	ヒ	235	♣
12	*(CLS)	44	,	76	L	108	l	140	▋	172	ャ	204	フ	236	●
13	*(RET)	45	—	77	M	109	m	141	▊	173	ュ	205	ヘ	237	○
14		46	.	78	N	110	n	142	▉	174	ョ	206	ホ	238	／
15		47	/	79	O	111	o	143	┼	175	ッ	207	マ	239	＼
16		48	0	80	P	112	p	144	┴	176	ー	208	ミ	240	╳
17	*(DEL)	49	1	81	Q	113	q	145	┬	177	ア	209	ム	241	円
18	*(INS)	50	2	82	R	114	r	146	┤	178	イ	210	メ	242	年
19		51	3	83	S	115	s	147	├	179	ウ	211	モ	243	月
20		52	4	84	T	116	t	148	▔	180	エ	212	ヤ	244	日
21		53	5	85	U	117	u	149	─	181	オ	213	ユ	245	時
22	*(ANS)	54	6	86	V	118	v	150	│	182	カ	214	ヨ	246	分
23	*(ENT)	55	7	87	W	119	w	151	▕	183	キ	215	ラ	247	秒
24	*(▲)	56	8	88	X	120	x	152	┌	184	ク	216	リ	248	〒
25		57	9	89	Y	121	y	153	┐	185	ケ	217	ル	249	市
26		58	:	90	Z	122	z	154	└	186	コ	218	レ	250	区
27		59	;	91	[	123	{	155	┘	187	サ	219	ロ	251	町
28	*(CSR>)	60	<	92	¥	124	¦	156	╭	188	シ	220	ワ	252	村
29	*(CSR<)	61	=	93	]	125	}	157	╮	189	ス	221	ン	253	人
30	*(CSR^)	62	>	94	^	126	~	158	╰	190	セ	222	゛	254	▩
31	*(CSRv)	63	?	95	_	127		159	╯	191	ソ	223	゜	255	

* (RET) functions as the function code for a line change.

SPECIFICATIONS

■Type

PB-700

■Fundamental calculation functions

Negative numbers, exponentials, parenthetical addition/subtraction/multiplication/division (with priority sequence judgement function — true algebraic logic).

■Built-in functions

Trigonometric/inverse trigonometric functions (angle units — degree/radian/gradient), logarithmic/exponential functions, square roots, powers, conversion to integer, deletion of integer portion, absolute value, symbolization, rounding, random numbers, π.

■Function digit capacity

	Input range	Output accuracy
$\sin x$, $\cos x$, $\tan x$	$\lvert x\rvert < 5440°$ ($30\ \pi$ rad, 6000 gra)	10th digit ±1
$\sin^{-1}x$, $\cos^{-1}x$	$\lvert x\rvert \leqq 1$	— " —
$\tan^{-1}x$	$\lvert x\rvert < 10^{100}$	— " —
$\log x$, $\ln x$	$x > 0$	— " —
e^x	$x \leqq 230.2585$	— " —
$\sqrt{x}$	$x \geqq 0$	— " —
x^y ($x \uparrow y$)	When $x < 0$, y is a natural number.	— " —

NOTE:
However, the output accuracy may change when a TAN calculation is performed in the following range.
DEG: $\lvert x\rvert = 90\ (2_{n-1})$
RAD: $\lvert x\rvert = \pi/2\ (2_{n-1})$
GRAD: $\lvert x\rvert = 100\ (2_{n-1})$
(n is an integer.)

■Commands

CONT, DELETE, EDIT, LIST, LLIST, LOAD, NEW, PASS, PROG, RUN, SAVE, SYSTEM, TROFF, TRON, VERIFY.
ANGLE, BEEP, CHAIN, CLEAR, CLS, DATA, DIM, DRAW, DRAWC, END, ERASE, FOR-TO-STEP, NEXT, GET, GOSUB, GOTO, IF-THEN-ELSE, INPUT, LET, LOCATE, LPRINT, PRINT, PUT, READ, REM, RESTORE, RETURN, STOP.

■Character functions

ASC, CHR$, LEFT$, LEN, MID$, RIGHT$, STR$, VAL.

■Other functions

INKEY$, POINT, TAB, USING.

■Calculation range

$\pm 1 \times 10^{-99} \sim \pm 9.999999999 \times 10^{99}$
(Internal calculation uses 12 digits of mantissa.)

■Program system

Stored program system

■Program language

BASIC

■Memory capacity for programs

Standard 4K bytes, expandable up to 16K bytes.

■Number of program areas

Maximum 10 (P0 — P9)

■Number of stacks

Subroutine	12 levels
FOR-NEXT loop	6 levels
Numerical values	8 levels
Operators	20 levels

■Display system

Liquid crystal display (20 digits x 4 lines)

■Display elements

32 x 160 dots (20 x 4 characters)

■Display contents

10 digit mantissa + 2 digit exponent

■Main component

LSI

■Power consumption

0.1W

■Power source

Main: 4 AA size batteries.
Sub (for RAM backup): 1 lithium battery (CR1220).

■Battery life

Main: Approximately 100 hours on type SUM-3 (continuous operation).
Sub: (see page 12).

■Auto power off

Power is automatically turned off approximately 8 minutes after last operation.

■Ambient temperature range

0°C to 40°C (32°F to 104°F)

■Dimensions

23mmH x 200mmW x 88mmD (⅞"H x 7⅞"W x 3½"D)

■Weight

315 g (11.1 oz) including batteries.

INDEX

A

ABS 235
Absolute value 235
A command 113
ACS 227
ANGLE 164, 223
Array variable 62
ASC 248
ASCII code 248
ASN 227
Assignment 33
ATN 227
Auto power off 13

B

Backup battery 12
Base of a natural logarithm 230
BASIC 30
B command 117, 118
BEEP 165
Built-in functions 309

C

Calculation methods 22
Calculation precision 21
Calculation priority sequence 21
CAPS 18, 19
C command 113, 120
CHAIN 166
Character array variable 57, 74
Character code 321
Character coordinates 26
Character mode specification 108
Character registered variable 25
Character string format specification 269
Character variable 57, 200
CHR$ 250
CLEAR 168
CLS 51, 169
Colon (:) 49
Comment statement 41
Common logarithm 230
Conditional expression 42, 129, 194
Conditional jump 194
CONT 136
Control variables 73
COS 225

D

DATA 212
D command 111
Debug 36
DEGREE 164, 224
DELETE 138
Detail flowcharts 84
DIM 74, 170
Dimension 59
Direct mode 18
Display contrast control 15
Display of number of digits 64
Dot 87
Dot pattern 95
DRAW 87, 175
DRAWC 87, 175

E

E 308
EDIT 141
Editing key 20, 141
END 43, 178
Enter key 15, 34
ERASE 179
EXP 233

F

F command 121
File formats 161
File name 129
Final value 180
Fixed variables 24
Flowchart 82
FOR~TO~STEP/NEXT 54, 180
Format character string 267
FRAC 239

G

G command 119
GET 185
GOSUB 55, 188
GOTO 43, 192
GRADIENT 164, 224
Graphic coordinates 26, 87
Graphic mode specification 108

H

Half-precision 23, 65
H command 116

I

I command 109
IF~THEN~ELSE 42, 194
Increment 180
Initialization 40, 44, 70
INKEY$ 262
INPUT 38, 41, 197
INT 237
Inverse trigonometric 227

J

J command 114
Jump 42, 43, 188, 192

L

L command 117
LEFT$ 257
LEN 261
LET 203
LGT 230
LIST 145
LLIST 145
LOAD 149
LOCATE 204
LOG 230
Logic flowcharts 84
Loop 180
LPRINT 108, 205

M

Main power source 12
Main program 188
Main routine 55
M command 119
Message 39
MID$ 259
MOD 308
Multistatement 49

N

Natural logarithm 230
N command 126
Nesting 180, 189
NEW 153
NEW ALL 153
Null-string 172
Number of bytes used 27
Numerical array 62
Numerical array variable 62
Numerical format specification 268
Numerical registered variable 25
Numerical variable 24, 57, 200

O

O command 116
One-dimensional arrays 60

P

PASS 155
PASS command release 156

Password 151, 155
P command 121
PI 245
Plotter command 109
Plotter-printer with cassette interface (FA-10) 86
POINT 87, 103, 271
PRINT 38, 43, 205
PROG 157
Program areas 35
Program modification 47
PUT 210

Q

Q command 124

R

RADIAN 164, 224
RAM expansion pack 14
Random number 246
R command 125
READ 212
Registered variable 25, 198
Relational operators 23
REM 217
RESTORE 212
RETURN 188
Return key 15, 34
RIGHT$ 258
RND 246
ROUND 243
RUN 158

S

SAVE 159
S command 122
Screen display control 48
Semicolon (;) 41, 43
SGN 241
Shift mode 19
SIN 222
Single-precision 23, 64
SQR 229
Square root 229
STOP 218
STR$ 255
Subroutine 55, 188
SYSTEM 50, 162

T

TAB 48, 265
TAN 226
T command 109
Ten keys 31
Trace mode 220
TROFF 220
TRON 220
Two-dimensional array 61

U

Unconditional jump 192
USING 49, 267

V

VAL 252
VERIFY 163

X

X command 116

Y

Y command 123

Z

Z command 123